НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

На правах рукопису
УДК 004.056.53

До захисту допущено
В. о. завідувача кафедри ММСА
О. Л. Тимощук
«___» _____ 2019 р.

Магістерська дисертація

на здобуття ступеня магістра за спеціальністю 122 Комп'ютерні науки
спеціалізація Системи штучного інтелекту
на тему: «Система автоматизованого розгортання моніторингу із застосуванням експертних систем»

Виконав:
студент ІІ курсу, групи КА-83мп
Шевчук Назар Миколайович _____

Керівник: доцент кафедри ММСА
к.т.н., доц. Дідковська М. В. _____

Рецензент: доцент кафедри ПЗКС
к.т.н., доц. Заболотня Т. М. _____

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів
без відповідних посилань
Студент _____

Київ
2019

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

Інститут/факультет ННК «Інститут прикладного системного аналізу»
(повна назва)

Кафедра Математичних методів системного аналізу
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою
Спеціальність (спеціалізація) – 122 «Комп'ютерні науки», освітня програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ММСА
_____ О. Л. Тимощук
«___»_____2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту Шевчуку Назару Миколйовичу

1. Тема дисертації: «Система автоматизованого розгортання моніторингу із застосуванням експертних систем», науковий керівник дисертації Дідковська Марина Віталіївна, к.т.н., доцент, затверджені наказом по університету від 08 листопада 2019 р. № 3862-с.

2. Термін подання студентом дисертації: 13 грудня 2019 року.

3. Об'єкт дослідження: засоби управління конфігурацій, система моніторингу побудована із застосуванням експертних систем.

4. Предмет дослідження: ОС сімейства Linux, система збору метрик Prometheus Exporter, система збереження метрик TSDB Prometheus, система візуалізації метрик Grafana, методи розробки експертних систем, засіб управління конфігурації Ansible.

5. Перелік завдань, які потрібно розробити:
1) дослідити особливості розробки та побудови експертних систем;
2) дослідити особливості налаштування та імплементації систем моніторингу;
3) дослідити засоби управління конфігурації;

4) опираючись на проведені дослідження розробити сценарій автоматизованого розгортання експертної системи на основі інструментів моніторингу;

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:

1) огляд архітектури експертної системи
2) огляд архітектури та принципів роботи системи моніторингу
3) принципи роботи засобів управління конфігурації
4) презентація

7. Дата видачі завдання: 05 вересня 2019 року.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	Отримання завдання	05.09.2019	Виконано
2	Аналіз існуючих рішень та вибір варіанту розробки	20.09.2019	Виконано
3	Розробка актуальної структури системи	27.09.2019	Виконано
4	Розробка плану тестування	31.09.2019	Виконано
5	Побудова системи	05.10.2019	Виконано
6	Розробка опису системи	23.10.2019	Виконано
7	Тестування системи та аналіз результатів	25.10.2019	Виконано
8	Розробка стртап-проекту	28.10.2019	Виконано
9	Оформлення дисертації	10.12.2019	Виконано
10	Подача матеріалів до захисту	18.12.2019	Виконано

Студент Н. М. Шевчук

Науковий керівник дисертації М. В. Дідковська

РЕФЕРАТ

Магістерська дисертація: 114 с., 36 рис., 26 табл., 15 джерел.

Об’єкт дослідження – засоби управління конфігурацій, система моніторингу побудована із застосуванням експертних систем

Предмет дослідження – ОС сімейства Linux, система збору метрик Prometheus Exporter, система збереження метрик TSDB Prometheus, система візуалізації метрик Grafana, методи розробки експертних систем, засіб управління конфігурації Ansible.

Методи дослідження – для побудови експертної системи використані інтрументи моніторингу, а також технолої виявлення аномалій з використанням засобів управління конфігурацій.

Мета дослідження – розробити систему автоматизованого розгортання моніторингу із застосуванням експертних систем.

Актуальність дослідження – у зв’язку з гострою потребою відстеження стану IT-інфраструктури, а також виявлення та запобігання збоїв її компенентів технології виявлення аномалій є дуже важливими.

Наукова новизна отриманих результатів заключається в наступному:

1. Дано пояснення використанню сучасних засобів управління конфігурацій.

2. Обгрунтовано застосування системи моніторингу Prometheus та Grafana при потребі виявлення аномалій в продуктовому середовищі.

3. Запропоноване рішення систеи моніторингу із застосуванням експертних систем.

Практичними результатами роботи є розроблений сценарій автоматизованого розгортання системи моніторингу, описані параметри конфігурації такої системи, що дозволяють виявляти аномалії, адаптуючись до змін поведінки кінцевого користувача IT-сервісу.

МОНІТОРИНГ, ЕКСПЕРТНІ СИСТЕМИ, ВИЯВЛЕННЯ АНОМАЛІЙ, IT-ІНФРАСТРУКТУРА, УПРАВЛІННЯ КОНФІГУРАЦІЇ, PROMETHEUS, GRAFANA, ANSIBLE

ABSTRACT

Master`s thesis: 114 p., 36 fig., 26 tabl., 15 sources.

The object of research – expert system based on monitoring tools.

Subject of research – Linux family OS, Prometheus Exporter metric collection system, TSDB Prometheus metric storage system, Grafana metric visualization system, expert systems development methods, Ansible configuration management tool.

Research methods – monitoring tools, as well as anomaly detection technologies using configuration management tools were used to build the expert system.

The purpose of the research – to develop a system of automated deployment of monitoring using expert systems.

Relevance of the research – in view of the urgent need to monitor the state of the IT infrastructure, as well as to detect and prevent the failure of its components, anomaly detection technology is very important.

The scientific novelty of the obtained results is the following:

1. The use of modern configuration management tools is explained.

2. The use of the Prometheus and Grafana monitoring system in case of anomalies detection in the product environment is justified.

3. The proposed solution of the monitoring system with the use of expert systems.

Results of the research – an automated deployment system was built using expert systems. During the research, it was found that anomaly detection helps to prevent the occurrence of failures of the IT-infrastructure components, as well as to establish the origin of these failures. This, in turn, is positive for the stability of the entire IT-system and reduces the financial and resource costs of restoring system performance.

The practical results of the work are developed a scenario of automated deployment of the monitoring system, described the configuration parameters of such a system, allowing to detect anomalies, adapting to changes in the behavior of the end user of the IT service.

MONITORING, EXPERT SYSTEMS, ANOMALY DETECTION, IT-INFRASTRUCTURE, CONFIGURATION MANAGEMENT, PROMETHEUS, GRAFANA, ANSIBLE

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Agile	–	методологія ведення розробки
ІС	–	Інформаційні Системи
УК	–	Управління Конфігурації
JSON	–	текстовий формат обміну даними
HR	–	Human resources
SLA	–	Service Level Agreement
SLO	–	Sservice-Level Objective
Postmortem	–	аналіз проблеми та кроки по її запобіганню/усуненню
SaaS	–	Software as a Service
HTTP(S)	–	протокол надсилання даних
QPS	–	Query Per Second
SNMP	–	Simple Network Management Protocol
AI	–	Artificial Intelligence
API	–	Application Programming Interface
ML	–	Machine Learning
CM	–	Configuration Management
SSH	–	Secure Shell
ОС	–	Операційна Система
YAML	–	текстовий формат обміну даними
IaaC	–	Infrastructure as a Code
DevOps	–	Development Operations
DSL	–	Domain Specific Languages
z-score	–	міра відносного відхилення від стандартного значення
RPS	–	Request Per Second

ВСТУП

В епоху тотальної діджиталізації величезні фінансові ресурси IT-компаній різних категорій (від стартап-проектів до гігантських корпорацій) витрачаються на розробку. При цьому хід ведення розробки потрібно постійно відслідковувати, а існуючі рішення - підтримувати. Як і прийняття будь-яких змін в проекті, так і підтримка створеного програмного забезпечення вимагає проведення глибокого аналізу та відслідковування величезної кількості метрик, що в свою чергу потребує впровадження новітніх технологій, збирання та зберігання гігантських масивів даних. Адже, саме обробка даних про стан інформаційних систем та різного роду сервісів дозволяє виконувати якісну підтримку та обслуговування IT-інфраструктури компанії, в тому числі оновлення та доопрацювання кодової бази, оновлення серверного програмного забезпечення, заміну мережевого обладнання та впровадження технічних рішень в сфері безпеки. Компанії, що нехтують подібними бізнес-процесами мають значно більше шансів зазнати непоправної шкоди, призведення якої може сприяти втраті важливих даних та фінансових ресурсів.

Основне завдання обробки даних про стан IT-інфраструктури полягає у виявленні та запобіганню збоїв її компонентів. Частіше всього результат такої обробки повинен показувати працездатність відповідних компонентів інфраструктури та гарантувати швидке встановлення джерела збою у разі його виникнення. Наприклад, ми маємо набір даних, що відображає послідовне систематичне фіксування стану системи. Встановленням деяких простих правил ми можемо відслідкувати підвищення рівня навантаження системи або використання ресурсів. Але що, якщо така поведінка системи закономірна? Виникає питання класифікації закономірності реакції системи на зовнішні чинники, такі як: кількість та тип запитів, а також внутрішніх: реалізації алгоритмів, структур даних та архітектури програмного забезпечення. В таких випадках жорстко постає задача виявлення аномалій.

Виявлення аномалії - це процес ідентифікації несподіваних елементів або подій у наборах даних, які відрізняються від норми.

Враховуючи поставлену задачу, варто відзначити, що експертні системи, окрім виконання обчислювальних операцій, здатні формувати певні висновки, базуючись на тих знаннях, якими вони володіють. Крім того, що основна кількість даних, що використовується для виявлення та запобігання збоїв, чи забезпечення безпеки генерується завдяки моніторингу, моніторинг в свою чергу гарантує сповіщення відповідальної особи про актуальний стан IT-інфраструктури або її компонентів. Моніторингом називають систему постійного відстеження процесів в середовищі розробки певних інформаційних систем. Результатом моніторингу є різні показники швидкості відгуку системи та показники стану самої системи, на що можуть впливати мережеві збої, збої різних програмних компонентів, а також наявність дефектів в коді. Інструменти моніторингу фіксують доступність компонентів IT-інфраструктури, збирають показники в реальному часі і здійснюють аналіз даних. Завдяки впровадженню системи моніторингу в компаніях стає можливим контролювати розгалужену IT-інфраструктуру з однієї точки. Реагуючи на потенційні небезпеки, система моніторингу допомагає виявити збої в інфраструктурі ще до виникнення інцидентів, що в свою чергу підвищує продуктивність праці і загальну ефективність IT-інфраструктури. Система моніторингу дозволяє контролювати безперебійність та правильність роботи різних інформаційних систем, таких як: інформаційна система аеропорту, банку, міжнародної біржі, інформаційних систем, що використовуються для підтримки державної безпеки і так далі.

Оскільки темп ведення розробки постійно зростає, а більшість компаній ведуть розробку та дослідження по методології Agile то і підтримка та імплементація нових рішень вимагає автоматизації. Задля економії ресурсів IT-підрозділу компанії використовуються практики безперервної інтеграції коду та автоматизованого розгортання і оновлення компонентів; практики автоматизованого тестування, а також практики централізованого керування конфігураціями та компонентами інфраструктури. Таким чином саме завдяки

генерації відповідальності зацікавлених сторін, зниженню ризиків та короткому циклу зворотнього зв’язку досягається ефективність розробки та підтримки інформаційних систем.

В цій роботі буде запропоноване рішення для автоматизації розгортання експертної системи на основі інструментів моніторингу за допомогою засобу управління конфігураціями. Застосування саме експертних систем в сучасних системах моніторингу дозволить ефективно виявляти незакономірну роботу системи за запобігати збоям, які можуть бути викликані різними чинниками.

Зокрема, завдяки використанню засобів УК досягається:

1) аварійне відновлення;
2) висока надійність системи;
3) легке масштабування;

При втраті працездатності інформаційної системи чи виходу з ладу певних компонентів, засоби УК дозволяють з найменшими затратами ресурсів відновити коректну роботу ІС, за потреби провести централізоване оновлення інфраструктури, що може складатись з тисячі та десятків тисяч серверів.

Термін "надійність системи" відноситься до того, як часто працює система. Частою причиною “простоїв” є погані розгортання, які можуть бути спричинені різницею у середовищах розробки. При правильному керуванні конфігурацією тестові середовища можуть імітувати продуктові, тож є менший шанс виникнення дефектів та втрати працездатності.

Управління конфігурацією гарантує, що стан системи правильний, тож коли необхідно зробити розгортання або імплементацію достатньо виконання декількох команд, замість того, щоб розпочинати налаштування заново.

Таким чином, основною ціллю даної роботи є побудова системи моніторингу на базі експертних систем, що здатна виявляти аномалії, а також автоматизація розгортання системи на серверах сімейства ОС Linux.

Для досягання цих цілей вирішені наступні задачі:

1. Провести аналіз існуючих рішень управління конфігурації для забезпечення автоматизованого розгортання моніторингової системи.

2. Провести аналіз існуючих рішень в області моніторингу, які дозволять виконувати якісне виявлення аномалій.
3. Розробити архітектуру системи моніторингу із застосуванням експертних систем для можливості виявлення аномалій в продуктовому середовищі.

Об’єктом дослідження є засоби управління конфігурацій, система моніторингу побудована із застосуванням експертних систем.

Предметом дослідження є ОС сімейства Linux, система збору метрик Prometheus Exporter, система збереження метрик TSDB Prometheus, система візуалізації метрик Grafana, методи розробки експертних систем, засіб управління конфігурації Ansible.

Наукова новизна отриманих результатів заключається в наступному:

1. Дано пояснення використанню сучасних засобів управління конфігурацій.

2. Обгрунтовано застосування системи моніторингу Prometheus та Grafana при потребі виявлення аномалій в продуктовому середовищі.

3. Запропоноване рішення систеи моніторингу із застосуванням експертних систем.

Практичними результатами роботи є розроблений сценарій автоматизованого розгортання системи моніторингу, описані параметри конфігурації такої системи, що дозволяють виявляти аномалії, адаптуючись до змін поведінки кінцевого користувача IT-сервісу.

РОЗДІЛ 1 ОСНОВИ ТА ПРИНЦИПИ РОБОТИ ЕКСПЕРТНИХ СИСТЕМ

Експертна система визначається як інтерактивна та надійна комп'ютерна система прийняття рішень, яка використовує як факти, так і евристику для вирішення складних проблем прийняття рішень. Вона розглядається на найвищому рівні людського інтелекту та експертизи. Система представляється як комп’ютерний додаток, який вирішує найскладніші проблеми в певній галузі.

Експертна система може вирішити багато питань, які, як правило, потребують втручання людини-експерта. Система заснована на знаннях, отриманих від експерта. Вона також здатна висловлювати міркування про певну область знань. Експертні системи були попередником сучасних систем штучного інтелекту, глибокого навчання та машинного навчання.

1.1 Приклади застосування експертних систем – додатків

Далі наведено приклади застосування експертних систем:

- управління інформацією;
- лікарні та медичні установи;
- допомога служби підтримки;
- оцінка ефективності працівників;
- аналіз позики;
- виявлення вірусів;
- корисно для відновлення та обслуговування проектів;
- оптимізація складу;
- планування та складання розкладу;
- моніторинг та контроль процесів;

- супервізор операцій;
- торгівля на фондовому ринку;
- розклад літаків та графіки вантажу.

1.2 Характеристика експертної системи

На рисунку 1.1 наведений приклад необхідности використання експертних систем:

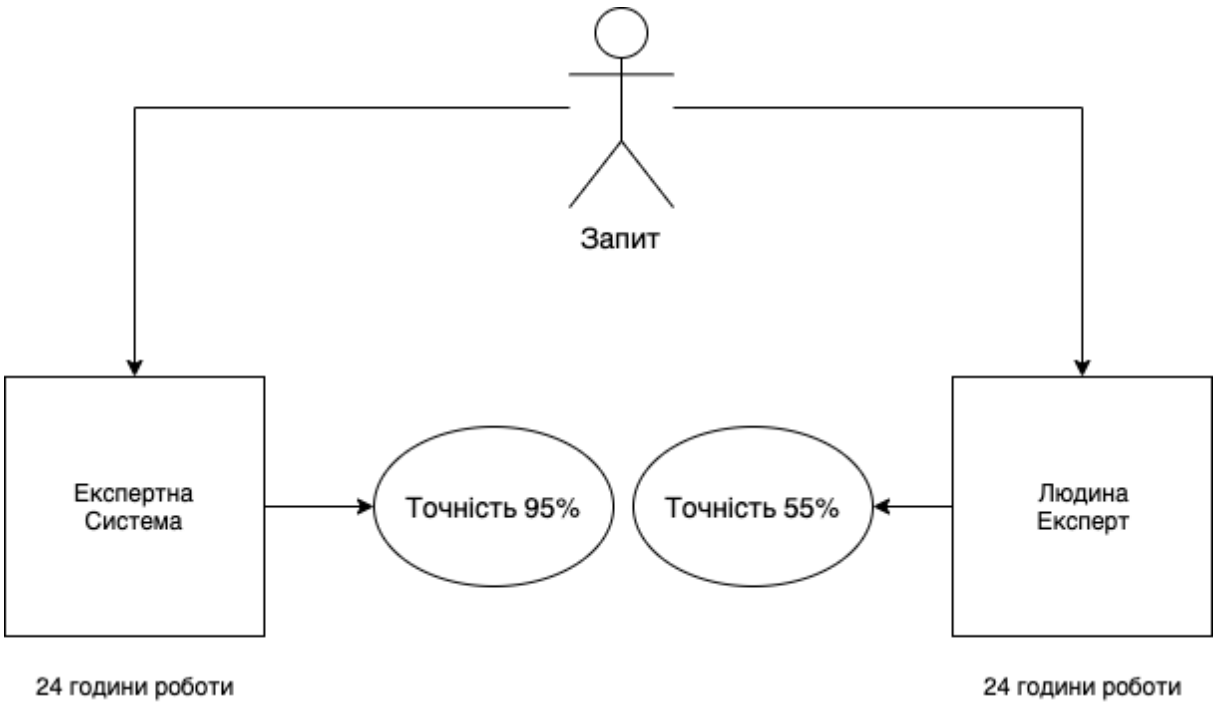

Рисунок. 1.1 - Необхідність використання експертних систем

Нижче наведені важливі характеристики експертної системи:

- Найвищий рівень експертизи: експертна система пропонує найвищий рівень проведення експертизи. Це забезпечує ефективність, точність та образне вирішення проблем.

– Миттєва реакція: експертна система взаємодіє з користувачем з дуже малою затримкою. Загальний час повинен бути меншим за час, який витрачає експерт, щоб отримати найбільш точне рішення тієї ж проблеми.
– Хороша надійність: експертна система повинна бути надійною, і вона має якнайменше помилятися.
– Ефективний механізм: експертна система повинна мати ефективний механізм класифікації наявних знань у ній.
– Здатна приймати важкі рішення та вирішувати складні проблеми: експертна система здатна виявляти складні проблеми та пропонувати рішення.

Взаємодія компонентів експертної системи наведено на рисунку 1.2:

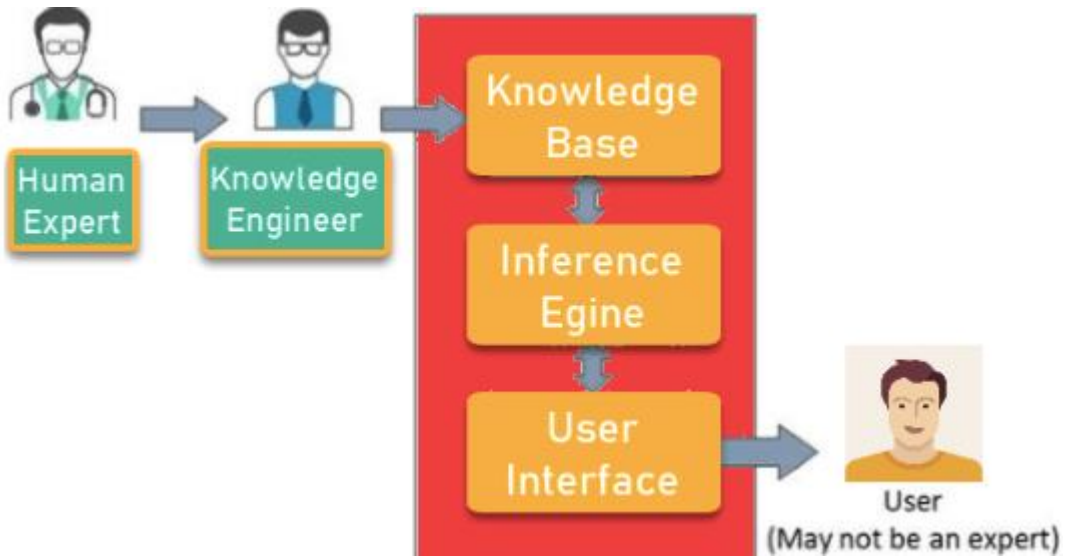

Рисунок. 1.2 - Компоненти експертних систем

Експертна система складається з таких компонентів:

1) інтерфейс користувача;
2) інтерпретатор;
3) база знань.

Користувацький інтерфейс - це інтерфейс, який допомагає користувачеві спілкуватися з експертною системою та отримувати результати класифікації даних.

Інтерпретатор - це мозок експертної системи. Інтерпертатор містить правила для вирішення конкретної проблеми. Він посилається на дані з бази знань. Він вибирає факти та правила, які слід застосувати, намагаючись відповісти на запит користувача. Він надає міркування про інформацію в базі знань. Це також допомагає при виведенні проблеми знайти рішення. Цей компонент також корисний для формулювання висновків.

База знань - це сховище фактів. Тут зберігаються всі знання про проблемну область. Це як великий контейнер знань, який отримують у різних експертів певної галузі. Таким чином, можна сказати, що ефективність експертної системи головним чином залежить від точних знань.

Роль керування за взаємодії з експертною систему переймає учасник. Перелік учасників експертної системи та описання їх ролей наведено в таблиці 1.1:

Таблиця 1.1 – Опис учасників розробки експертних систем

Учасник	Роль
Експерт з домену	Людина чи група, чий досвід та знання в певній предметній області використовуються для розробки експертної системи.
Інженер знань	Технічна особа, яка інтегрує знання в комп'ютерні системи
Кінцевий користувач	Людина або група людей, які використовують експертну систему, щоб отримати поради, які не надасть експерт

1.3 Інші ключові терміни, використовувані в експертних система

1.3.1 Факти та правила

Факт - це невелика частина важливої інформації. Факти самі по собі є дуже обмеженими. Правила мають важливе значення для вибору та застосування фактів до проблеми користувача.

1.3.2 Придбання знань

Термін отримання знань означає, як отримати необхідні знання про предметну область експертною системою. Весь процес починається з вилучення знань у людини-експерта, перетворення отриманих знань у правила та введення розроблених правил у базу знань. Приклад наведений на рисунку 1.3:

Рисунок. 1.3 - Процес отримання знань

1.4 Процес побудови експертних систем

Процес побудови експертних систем включає наступні кроки:

- визначення характеристики проблеми;
- інженер знань та експерт із домену (предметної області) працюють узгоджено для визначення проблеми;
- інженер знань перекладає знання на зрозумілу для комп'ютера мову. Він розробляє інтерпретатор, структуру міркувань, яка може використовувати знання при необхідності;
- експерт знань також визначає, як інтегрувати використання невизначених знань у процесі міркування та який тип пояснення був би корисним.

1.5 Порівняння експертної системи з іншими шляхами експертизи

1.5.1 Порівняння звичайної та експертної систем

Проведемо порівняльну характеристику звичайної та експертної систем, наведеної в таблиці 1.2:

Таблиця 1.2 – Порівняння звичайно та експертної систем

Звичайна система	Експертна система
Знання та обробка об’єднані в одне ціле.	База знань та механізм обробки є двома окремими компонентами.
Програма не допускає помилок (якщо тільки помилка в програмуванні).	Експертна система може помилитися.

Продовження таблиці 1.2

Система функціонує лише коли повномістю розрозблена.	Експертна система оптимізується на постійній основі та може бути запущена з невеликою кількістю правил.
Покрокове виконання за фіксованими алгоритмами потрібно.	Виконання виконується логічно та евристично.
Для цього потрібна повна інформація.	Він може бути функціональним з достатньою або недостатньою інформацією.

1.5.2 Порівняння людини - експерта та експертної системи

Порівняльну характеристика проведених експертиз людиною та експертною системою, наведеної в таблиці 1.3:

Таблиця 1.3 – Порівння результатів експертизи людини та експертної системи

Людина - експерт	Штучна система
Швидкопсувна	Постійна
Важко переноситься	Легко переноситься
Складний процес документації	Легко документується
Непередбачувана	Послідовна
Дорога	Фінансово ефективно

1.6 Переваги експертних систем

Перелік переваг експертної системи:

- покращує якість рішення;
- зменшує витрати на консультаційні експерти для вирішення проблем;
- забезпечує швидке та ефективне вирішення проблем у вузькій галузі спеціалізації;
- може зібрати дефіцитний досвід і ефективно його використовувати;
- пропонує послідовну відповідь на проблему, що повторюється;
- підтримує значний рівень інформації;
- допомагає отримати швидкі та точні відповіді;
- правильне пояснення прийняття рішень;
- здатність вирішувати складні питання;
- експертні системи можуть стабільно працювати, не переживаючи емоційних напружень чи втоми.

1.7 ВИСНОВОК ДО РОЗДІЛУ 1

Експертна система - це інтерактивна та надійна комп'ютерна система прийняття рішень, яка використовує як факти, так і евристику для вирішення складної проблеми прийняття рішень.

Основними компонентами експертної системи є:

1) інтерфейс користувача;
2) інтерпретатор;
3) база знань.

Основними учасниками розробки експертних систем є:

1) експерт домену;

2) інженер знань;
3) кінцевий користувач.

Покращена якість рішення, зниження витрат, послідовність, надійність, швидкість - основні переваги експертної системи.

Експертна система не може давати творчих рішень але вартість підтримки такої системи значно менша ніж людини-експерта.

Експертна система може використовуватися в широких сферах застосування, таких як Фондовий ринок, облік складу, HR та інше.

РОЗДІЛ 2 ОГЛЯД ТА ЗАСТОСУВАННЯ ІНСТРУМЕНТІВ МОНІТОРИНГУ ІТ-ІНФРАСТРУКТУРИ

Моніторинг ІТ-інфраструктури - це розгортання вбудованої бази знань для автоматичного діагностування проблем продуктивності та доступності всіх компонентів інфраструктури. Повний стек моніторингу ІТ-інфраструктури включає:

- моніторинг обладнання - фізичний стан системи;
- моніторинг ОС - працездатність та використання ресурсів;
- моніторинг мережі - споживання пропускної здатності та помилки;
- моніторинг додатків - Продуктивність та доступність.

Оскільки ІТ-інфраструктури часто складаються з декількох локацій розгортання компонентів (приватні, громадські та гібридні хмарні сховища) перед моніторингом стоїть завдання швидко визначити і співвіднести проблеми, перш ніж вони вплинуть на кінцевих споживачів і в кінцевому підсумку на продуктивність організації.

Моніторинг ІТ-інфраструктури стає складнішим, оскільки компоненти мають властивість масштабування та рознесення по різним локаціям. Необхідність збору та аналізу великої кількості серверних даних вимагає автоматизації. Це дозволяє ІТ-персоналу витрачати свої обмежені ресурси імплементацію важливих технічних рішень замість того, щоб постійно вирішувати проблеми, що виникають.

2.1 Дизайн моніторингу IT-інфраструктури

Перше запитання, на яке необхідно дати відповідь при виборі чи побудові системи моніторингу, це чи готові компанії, яких представляють керівники проектів, витрачати фінансові кошти за комерційне рішення? Зазвичай, це дозволяє зекономити час відкинувши заздалегідь неприйнятні рішення.

Нижче також наведено перелік запитань, відповіді на котрі допоможуть безболісно обрати системи моніторингу для подальшої імплементації:

1) Що має перебувати під моніторингом?
   - обладнання - “металеві” сервери та хмарна віртуалізація;
   - операційні системи – Windows, Unix та Linux;
   - мережа - пристрої з підтримкою SNMP;
   - програмні компоненти - бази даних, додатки, різноманітні сервіси.
2) Який зміст проблеми?
   - помилки обладнання та проблеми з доступністю;
   - недостатня забезпеченість ресурсами Windows та *Nix (процесор, пам'ять, накопичувач, ширина смуги пропускання мережі);
   - надмірне використання мережі та рівень помилок;
   - проблеми із застосуванням, визначені вбудованою базою знань.
3) Що робити, коли буде виявлена проблема?
   - за можливості необхідно автоматизувати команду або сценарій для усунення проблеми;
   - потрібно визначити пріоритети та налаштувати сповіщення про важливі події за допомогою текстових повідомлень або електронних сповіщень, наприклад, централізованих панелей моніторингу;
   - продуктивність / доступність інформаційної панелі на основі критеріїв, визначених користувачем.

4) Який тип репортингу має використовуватись?
   - аналітика - тенденції та моделі ефективності;
   - планування потенціалу - обмеження впливу на IT-інфраструктуру;
   - договір про рівень обслуговування (SLI/SLO) - базова поведінка у разі виявлення компонентів IT-інфраструктури, певні значення яких відхиляються від норми;
   - проблеми (postmortem) - зберігання історії виникнення подій.

2.2 Огляд популярних рішень

2.2.1 Nagios XI

Nagios - це, мабуть, найпопулярніший інструмент мережевого та інфраструктурного моніторингу, частково тому, що він також найстаріший (серед інструментів моніторингу, які досі використовуються). Це рішення на основі Linux, яке є дуже гнучким і потужним, оскільки ядро системи моніторингу можна розширити плагінами. Приклад користувацького інтерфейсу наведено на рисунку 2.1:

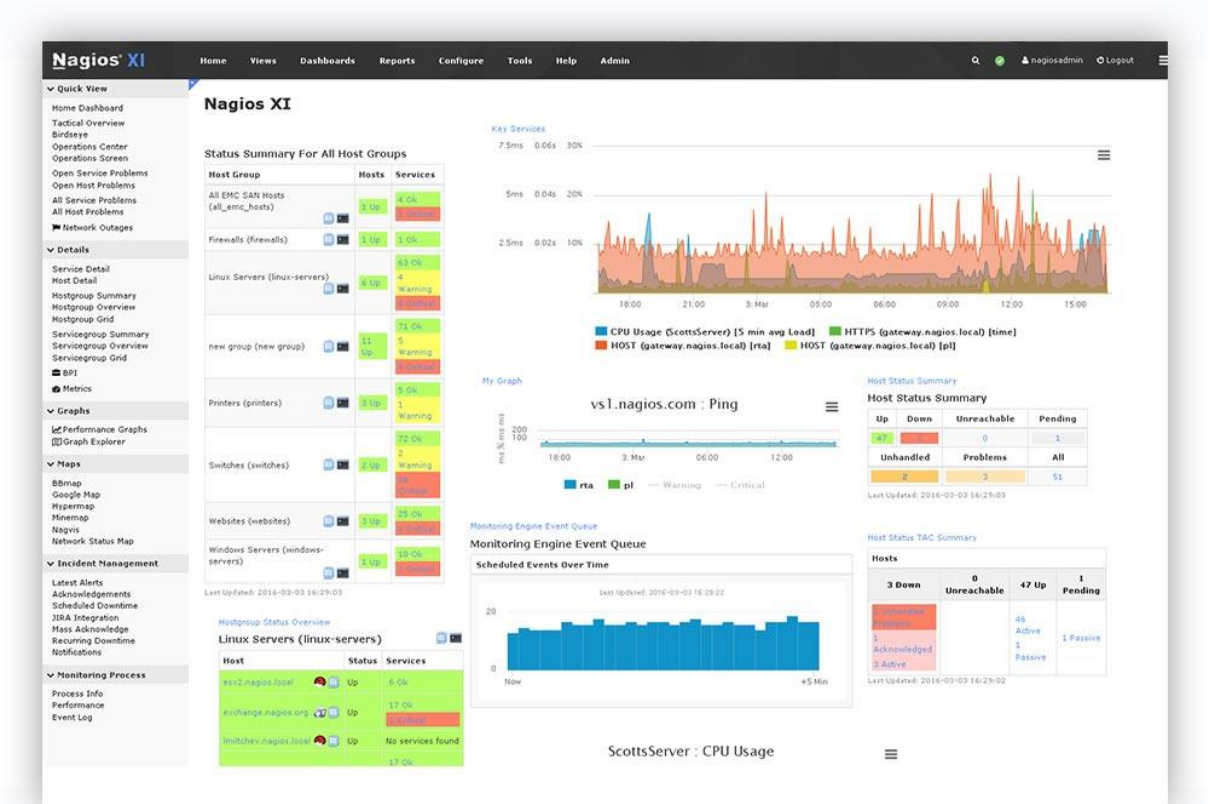

Рисунок. 2.1 - Приклад користувацького інтерфейсу Nagios XI

Nagios поставляється у двох варіантах: Nagios Core, який є безкоштовним та з відкритим кодом, та Nagios XI, який є платним корпоративним виданням. Nagios Core не підтримує досить багато цікавих функцій: звітність, текстова конфігурація, підтримка візуалізації графіків. Однак Nagios XI є значно розширеною версією Nagios Core, та за замовчуванням має багато можливостей, яких не вистачає в Nagios Core. Наведемо приклад деяких переваг Nagios XI перед безкоштовною версією:

- розширений веб-інтерфейс;
- авто-відкриття;
- графіки;
- сповіщення (SMS, електронна пошта);
- звітність;
- помічник конфігурації.

Nagios XI доступний у двох виданнях: Standard Edition та Enterprise Edition. Ліцензії на кожне видання доступні у трьох варіантах: на 100 вузлів, на 200 вузлів або на необмежену кількість вузлів. Найдешевша версія Nagios XI (Standard 100 Node Edition) коштує 1995 доларів.

2.2.2 Монітор сервера та додатків Solarwinds (SAM)

Приклад користувацького інтерфейсу Solarwinds SAM наведено на риску 2.2:

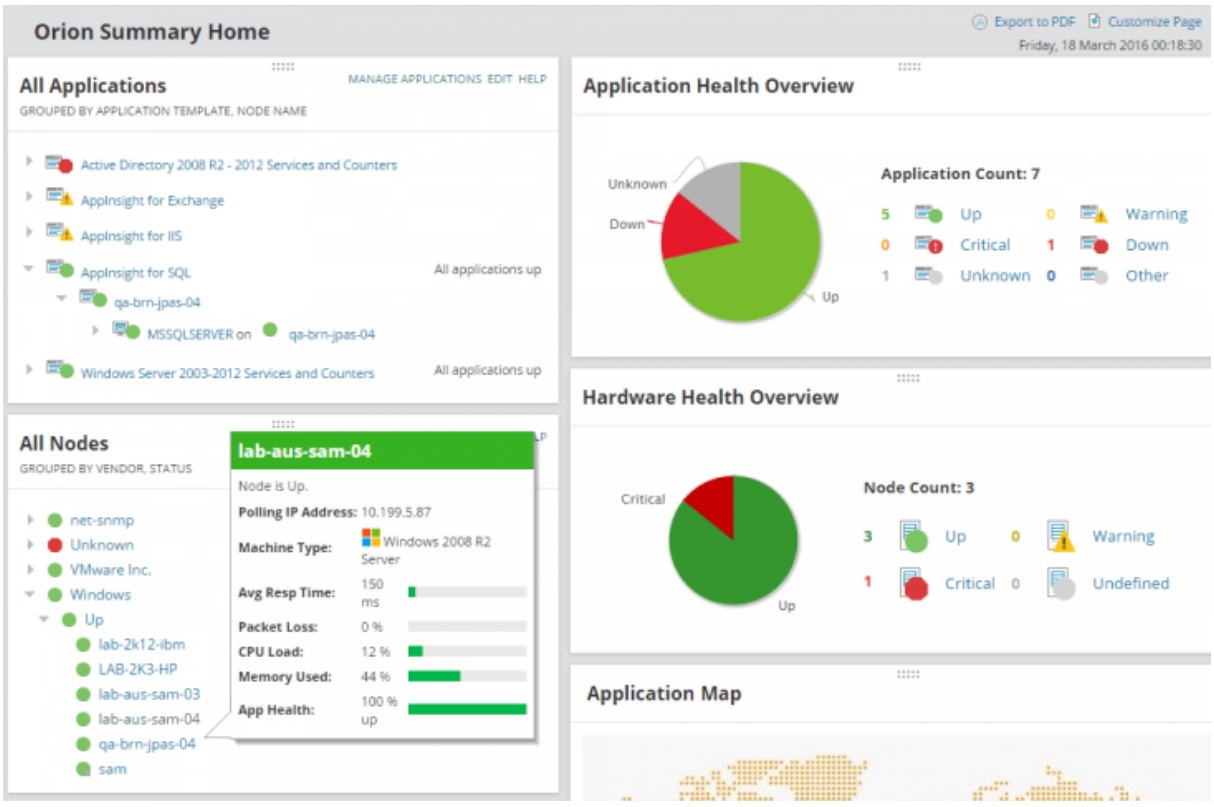

Рисунок. 2.2 - Приклад користувацького інтерфейсу Solarwinds SAM

Solarwinds SAM забезпечує глибоке розуміння серверів (наприклад, серверів IBM і Dell) та додатків (наприклад, Microsoft Exchange, IIS і Java). У інструменті є шаблони моніторингу, які допоможуть швидко налаштувати

потрібну конфігурацію. Шаблони також можна налаштувати для моніторингу користувацьких програм.

Solarwinds SAM доступний для безкоштовного 30-денного випробування, після чого ліцензії починаються від 2995 доларів за 150 моніторів.

2.2.3 ManageEngine OpManager

Приклад користувацького інтерфейсу Manage OpManage наведено на рисунку 2.3:

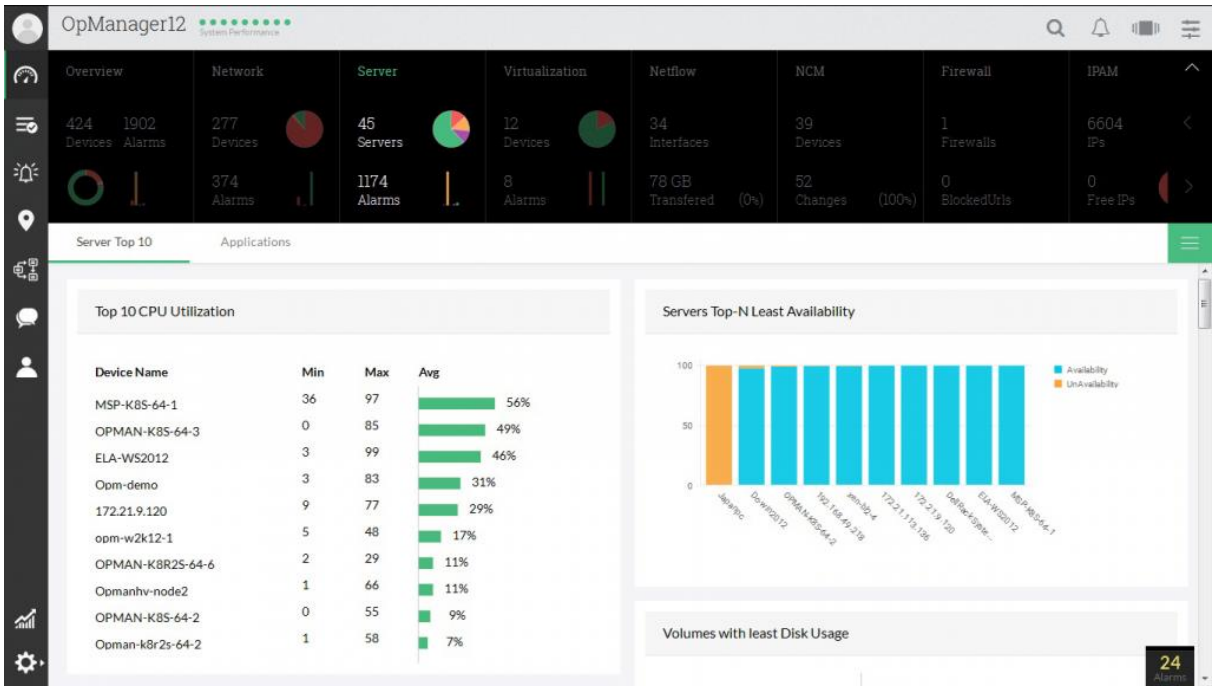

Рисунок. 2.3 - Приклад користувацького інтерфейсу ManageEngine OpManager

ManageEngine OpManager - це комплексне рішення для моніторингу IT-інфраструктури з простим у користуванні чуйним веб-інтерфейсом, як це видно з нашого огляду OpManager тут. Його можна встановити на ОС Windows або Linux та пропонує декілька функцій, таких як:

- моніторинг сервера;
- картування мережі;
- шаблони моніторингу;
- сповіщення (SMS, електронна пошта);
- звітність;
- управління мережевою конфігурацією;
- аналіз мережевого трафіку.

Хоча більшість цих функцій входять до базової версії (безкоштовна версія), деякі з них вимагають придбати окрему ліцензію. Наприклад, модуль управління конфігурацією мережі безкоштовний для двох пристроїв, після чого йому потрібна окрема ліцензія. Модуль аналізу мережевого трафіку також вимагає ліцензії після використання двох інтерфейсів.

ManageEngine OpManager має три типи ліцензій (крім безкоштовної 30-денної пробної версії): Essential, OpManager Plus та Enterprise. Ліцензія Essential починається від 595 доларів для 25 пристроїв.

Також існує також безкоштовне видання для моніторингу 10 вузлів, але з обмеженою функціональністю.

2.2.4 Zabbix

Zabbix - це безкоштовне відкрите джерело моніторингу з відкритим кодом. Він використовує модель клієнт-сервер, де сервер Zabbix отримує інформацію про моніторинг від агента Zabbix (хоча моніторинг без агентів також підтримується). Деякі функції, які надає Zabbix:

- конфігурація на основі веб-сторінок;
- моніторинг ефективності;
- моніторинг додатків;
- шаблони моніторингу для швидкого та простого налаштування;

– авто-відкриття;

– сповіщення;

– звітність.

Zabbix абсолютно безкоштовний - він не має платного корпоративного видання. Однак компанія, яка стоїть за Zabbix, пропонує платні служби підтримки, включаючи 5 рівнів технічної підтримки, інтеграції та консалтингові послуги. Приклад користувацького інтерфейсу Zabbix наведено на рисунку 2.4:

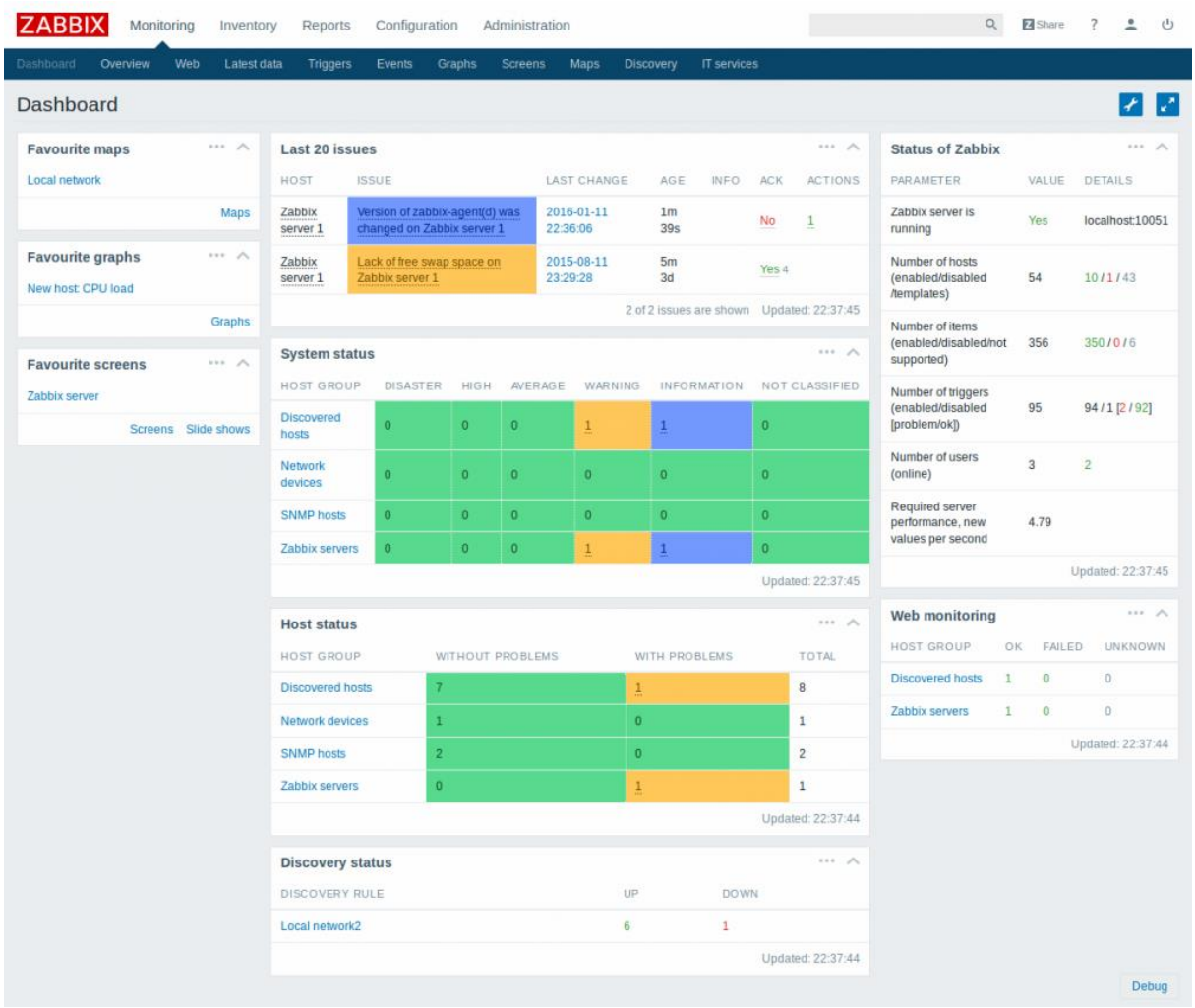

Рисунок. 2.4 - Приклад користувацького інтерфейсу Zabbix

2.2.5 Icinga

Приклад користувацького інтерфейсу Icinga наведено на рисунку 2.5:

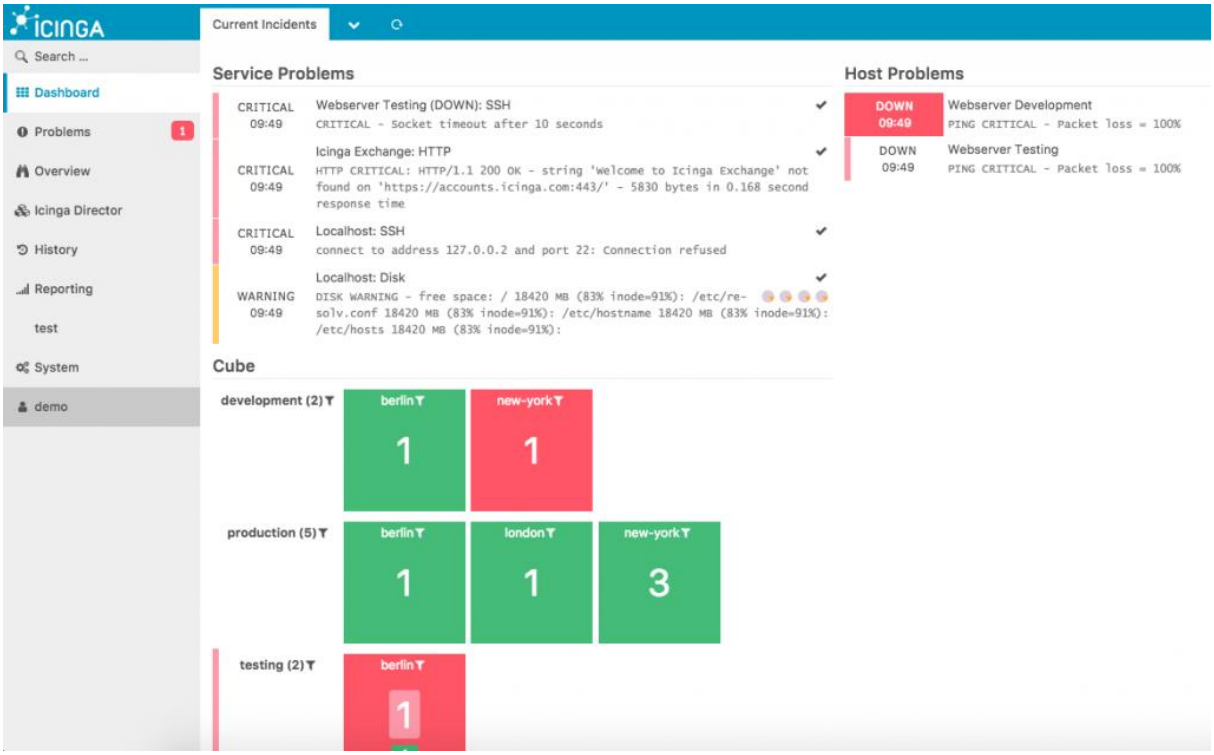

Рисунок. 2.5 - Приклад користувацького інтерфейсу Icinga

Icinga - це інструмент моніторингу мереж з відкритим вихідним кодом, який постачається у двох версіях: Icinga 1 та Icinga 2. Icinga надає багато функцій, включаючи:

- моніторинг ефективності;
- сповіщення;
- звітність;
- побудова графіків;
- розширюваність за допомогою плагінів.

Icinga 1 - це відгалудження Nagios Core з додатковою функціональністю, як підтримка більшої кількості баз даних, кращий веб-інтерфейс та легша

інтеграція плагінів. Незважаючи на те, що це відгалудження, Icinga 1 також сумісний з плагінами Nagios.

Icinga 2 - це повністю покращений Core, який має зручніший веб-інтерфейс. Конфігурація все ще здійснюється за допомогою текстових файлів, але Icinga 2 зменшує складність конфігурації. Icinga 2 була розроблена таким чином, щоб бути масштабованою, тому вона підтримує розподілений моніторинг.

Icinga повністю безкоштовна і не має платних видань. Однак, як і Zabbix, має послуги платної підтримки.

2.2.6 ELK (стек ElasticSearch, Logstash, Kibana)

Одним з найбільш використовуваних механізмів моніторингу та логування систем у більшості компанії є ELK Stack, який складається з наступних основних компонентів: Elasticsearch, Logstash та Kibana. Загалом ELK Stack використовується для розробки інструментів моніторингу за журналами (так званими логами). ELK Stack має перевагу в тому, що він може розгортатися як SaaS в більшості хмарних провайдерах.

Elastic стек - це набір інструментів з відкритим кодом, розроблений Elastics, який дозволяє збирати дані з будь-якого типу джерела та в будь-якому форматі для здійснення пошуку, аналізу та візуалізації даних у режимі реального часу.

Він складається з таких продуктів:

1) Elasticsearch: розповсюджена пошукова система, побудована на Apache Lucene. Він дозволяє індексувати та витягувати документи JSON (без жорсткої схеми) у різних форматах. Він заснований на Java і забезпечує доступний RESTFUL інтерфейс.

2) Logstash: це двигун, який дозволяє збирати дані з різних джерел, нормалізувати їх і поширювати. Спочатку оптимізовано для файлів-журналів, хоча він може читати дані з багатьох типів джерел.
3) Kibana: інструмент, що дозволяє візуалізувати та досліджувати в реальному часі велику кількість даних. За допомогою графічного зображення дозволяє легко візуалізувати складні набори даних.
4) Beats: це завантажувач даних, який встановлюється на сервери і функціонує як агент та надсилають оперативну інформацію в Elasticsearch.

Elastic-стек дозволяє отримувати дані з журнальних файлів за допомогою Logstash та зберігати їх в аналітичній пошуковій системі Elasticsearch. Крім того, це дозволяє виконувати візуалізацію, моніторинг та управління даними в режимі реального часу через веб-інтерфейс Kibana.

З іншого боку, пропонується включити ElastAlert, інструмент з відкритим вихідним кодом, інтегрованим з Elastic-стек і дозволяє виявити аномалії та невідповідності даних Elasticsearch, а також виконувати сповіщення.

Набір інструментів Elastic-стеку, а також ElastAlert дозволяє створити систему моніторингу для вилучення, зберігання та експлуатації даних процесів або систем, що підлягають моніторингу. Це дозволяє отримувати дані з журнальних файлів через Logstash та зберігати їх у системі пошуку та аналізу Elasticsearch. Крім того, це дозволяє візуалізацію, моніторинг та експлуатацію даних у режимі реального часу через Kibana та конфігурацію попереджень з ElastAlert.

Для зберігання даних у Elasticsearch Logstash використовує вихідний плагін Elasticsearch, який дозволяє надсилати дані через протокл HTTP та HTTP. Elasticsearch буде відповідальним за індексацію даних, отриманих із журналів, та можливість їх пошуку та аналізу. Схема обробки даних за допомогою ELK стеку наведена на рисунку 2.6:

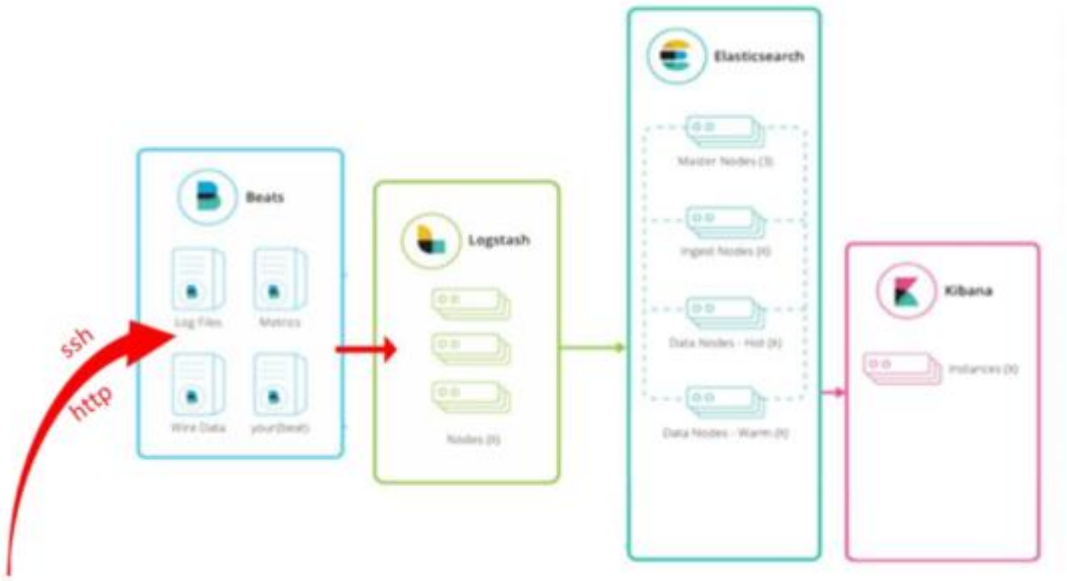

Рисунок. 2.6 - Схема обробки даних за допомогою Elastic стеку

2.2.7 Prometheus – Grafana

Prometheus - Grafana - це досить популярна комбінація інструментів для створення системи моніторингу.

Таке поєднання дозволяє дізнатись, наприклад, скільки запитів обробляються за певний проміжок часу (QPS), середній час відповіді та тим часом, скільки ресурсів (процесор, пам'ять, введення/виведення) використовується.

2.2.7.1 Prometheus: колектор метрик

Prometheus - система моніторингу систем та послуг, збирає показники із заздалегідь визначених цілей за допомогою Pull-моделі. Цілі можна знайти в службі Discovery або налаштувати вручну, щоб витягувати дані з кінцевих точок

системи, наприклад серверу API, балансувальника навантаження, різноматних SQL та NoSQL баз даних. На рисунку 2.7 наведена схема архітектури Prometheus:

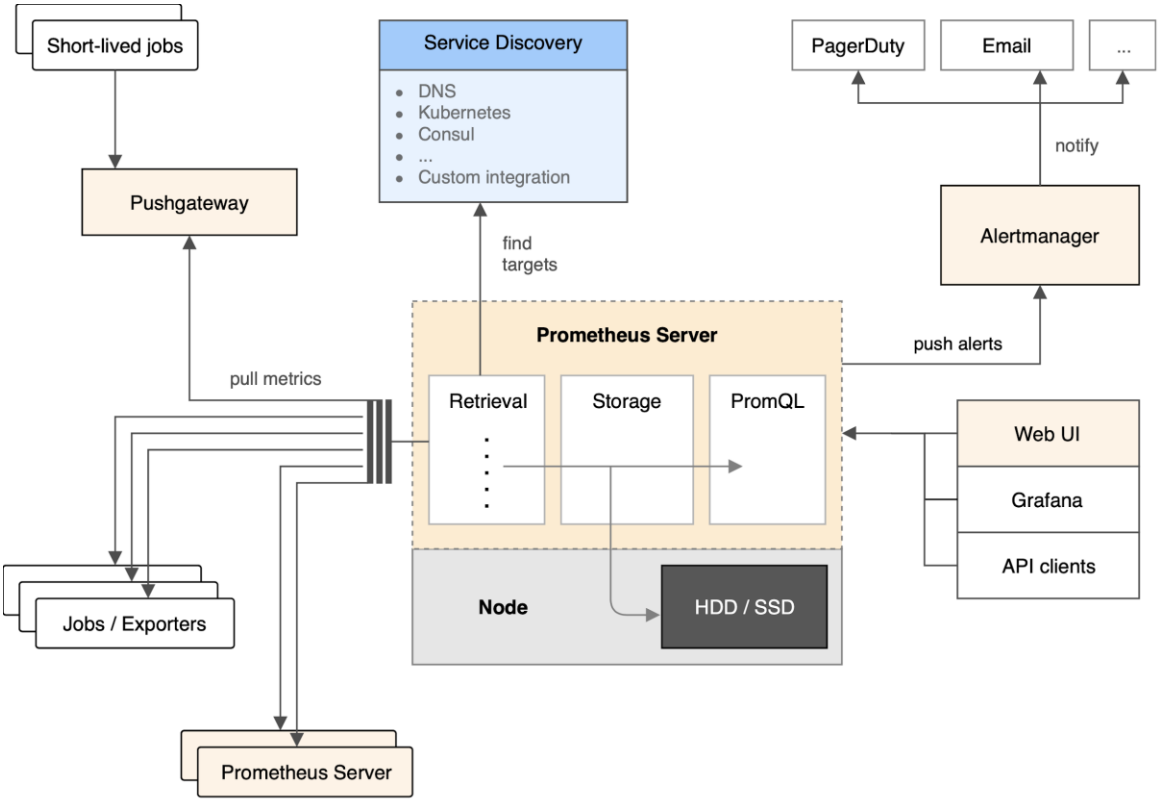

Рисунок. 2.7 - Огляд архітектури Prometheus (від GitHub)

2.2.7.2 Grafana: візуалізатор даних

Grafana підтримує безліч джерел даних, таких як: CloudWatch, Stackdriver, Elasticsearch, InfluxDB і, звичайно, Prometheus. Підключивши Grafana до джерел даних, можливо з легкістю створити інформаційну панель вручну або імпортувати різні інформаційні панелі, які доступні на офіційному веб-сайті.

Щоб створити систему моніторингу з Prometheus та Grafana, нам потрібні три компоненти:

1) Експортер метрик: зазвичай це бінарний файл, що збирає метрики локально та робить їх доступними для завантаження, періодично оновлюючи.
2) Prometheus Сервер: періодично витягує показники з експортерів / робочих місць.
3) Grafana: виконує запит на сервер Prometheus для створення візуальних інформаційних панелей.

   На рисунку 2.8 наведений приклад інформаційної панелі Grafana:

Рисунок. 2.8 - Приклад панелі відображення візуалізованих даних в Grafana

2.2.7.3 Експортер показників

По-перше, сервер Prometheus потребує імпорту метрик, щоб він міг збирати показники та генерувати базові графіки.

Таке програмне забезпечення часто має певний інтерфейс, через який можна отримати доступ до показників за допомогою протоколу HTTP. Це може

бути спеціальний формат, що вимагає спеціального розбору та обробки, як це потрібно для багатьох показників Linux, або встановленого стандарту, наприклад SNMP.

Експортер - це програмне забезпечення, що розгортається біля програми, від якої необхідно отримати показники. Він збирає потрібні дані з відповідної програми, перетворює їх у правильний формат, приймає запити від Prometheus, і, нарешті, у відповідь Prometheus всі зібрані дані. Можна розглядати експортера як невеликий проксі-сервер, який перетворює дані між метричним інтерфейсом програми та форматом експозиції Prometheus.

На відміну від прямого інструментарію, який ми використовували б для керування кодом, експортери використовують інший стиль інструментарію, відомий як користувацькі колектори або ConstMetrics.

Надзвичайно великою перевагою є розмір спільноти Prometheus, оскільки це програмне забезпечення з відкритим вихідним кодом. Скоріш за все потрібний експортер вже існує і може бути використаний з невеликими змінами у конфігурації. Якщо експортер не має можливісті надати потрібні показники – завжди є змога дописати функціонал експортеру самостійно, або ж відправити запит на добрацювання розробнику експортера.

2.3 Застосування лінійного прогнозування залишку фізичної пам’яті на диску

Розглянемо можливості Prometheus на прикладі проблеми нестачі фізичної пам’яті на диску. Заповнення диска небажане, оскільки це може блокувати операцію запису, в результаті чого багато програм та утиліт не справляються із тим, що не можуть внести зміни у файли. Стандартним способом захисту від цього є отримання сповіщень, коли диск заповнюється, в результаті чого відповідальний інженер має вирішити проблему, перш ніж диск заблокується на запис. Зазвичай це робиться на основі простих порогових

значень, таких як 80%, 90% або 10 ГБ, що залишилися. Це працює, коли на всіх серверах відсутні сплески та використання ресурсів є рівномірним, але не так добре, коли спостерігається дуже різкий ріст або зростання настільки швидке, що до моменту отримання сповіщення інженер вже не має змоги запобігти проблемі (наприклад, коли таких дисків десятки, а то і сотні).

Яким чином замість фіксованого порогового значення можна попередити заповнення диску за 4 години? Функція predict_linear(), що зображена на рискунку 2.9 дозволяє робити саме це. Вона використовує лінійну регресію протягом певного періоду часу, щоб передбачити, яким буде значення часових серій у майбутньому.

```
expr: predict_linear(node_filesystem_free{job="node"}[1h], 4 * 3600) < 0
```

Рисунок. 2.9 - Використання функції лінійного прогнозуванні в Prometheus

На рисунку 2.9 зображено приклад виразу, який викликає сповіщення менеджера попередження. “node_filesystem_free {job = 'node'} [1h]” отримує історію за попередню годину. Це передається до функції “predict_linear”, яка виконує прогнозування на 4 годин вперед, оскільки в годині є 3600 секунд. ” <0” - це фільтр, який повертає лише значення, менші за 0.

Таким чином, виконується проактивний моніторинг, який призначений для виявлення вичерпання системних ресурсів, таких як пам’ять або дисковий простір. Вартно зазначити, що немає нічого поганого в 100% завантаження процесора за умови, що це не означає відсутність продуктивності системи.

Якщо наявні деякі причини чому процесор не повинен досягати 100%, в такому разі краще отримувати сповіщення, якщо процесор затримався на 100% завантаження протягом певного часу.

Повертаючись до питання прогнозу, варто враховувати розмір вхідних даних. Адже прогнозування на основі 5 хвилин даних для наступних 5 годин

спричинить невірні сповіщення. Не рідкість програма, яка збільшує споживання на короткочасній основі. Більше того, навіть якщо форма використання пам’яті додатків є ідеальним кроком, функція predvi_linear() використовує лінійну регресію для обчислення швидкості у певний момент часу.

Таким чином краще використовувати більші проміжки часу для прогнозування, щоб попередити хибні сповіщення.

2.4 Застосування Prometheus для виявлення аномалій

Із збільшенням кількості метрик, що аналізуються за допомогою Prometheus, стає важче виявляти сигнали в межах шуму. Сучасний стан техніки представляється як графік показників на інформаційних панелях та сповіщення про порогові показники, що в даний час здійснюється. Тобто люди, які обслуговують систему, встановлюють ці пороги самостійно.

За допомогою підходу на основі AI з’являється можливість навчити моделі машинного навчання на історичних метричних даних для виконання прогнозування часових рядів. Справжні метричні значення можуть бути порівняні з прогнозами моделі. Якщо передбачуване значення сильно відрізняється від справжнього метричного значення, ми можемо повідомити про це як про аномальну поведінку.

Основні вимоги до системи виявлення аномалії Prometheus:

1) Prometheus метрики - важливі показники, які потрібно контролювати;
2) Grafana - інструмент візуалізації, призначений для створення графіків даних часових рядів Prometheus.

На рисунку 2.10 наведена приклад схеми поєднання цих двох інструментів в одну систему:

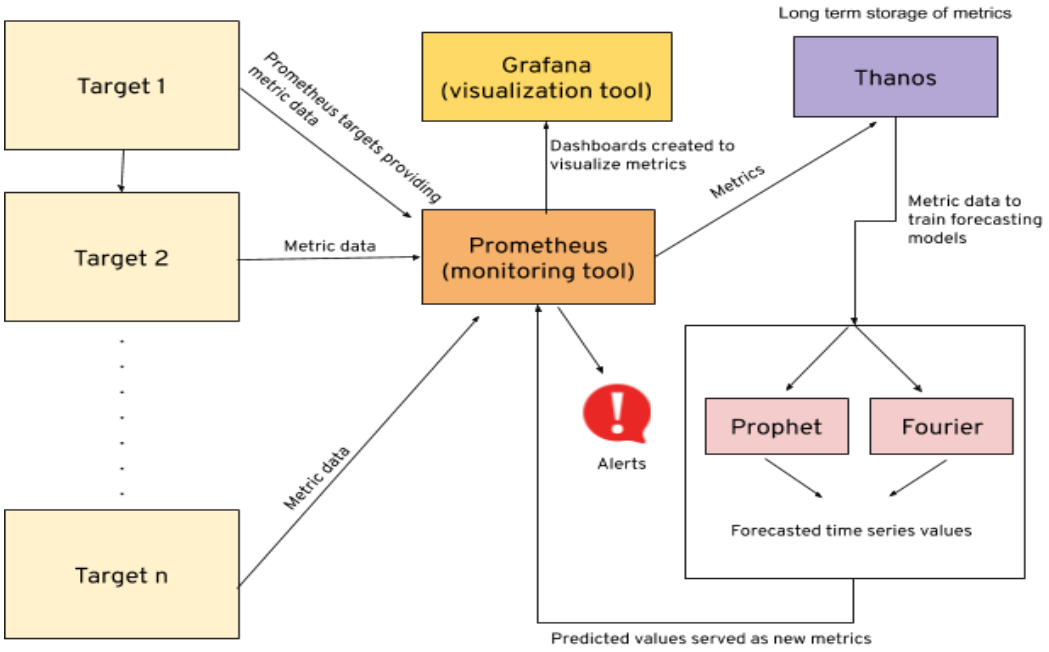

Рисунок. 2.10 - Приклад архітектури системи виявлення аномалій на базі Prometheus - Grafana

Реалізація системи виявлення аномалій полягає в наступному:

1) дані - показники часового ряду Prometheus, зібрані з заданих хостів / цілей;
2) моделі, що навчаються:
   - фур'є - використовується для відображення сигналів з часової області в частотну область. Він представляє дані періодичних часових рядів у вигляді суми синусоїдальних компонентів;
   - пророк – це модель пророків розроблена Facebook для прогнозування даних часових рядів на основі адитивної моделі, де нелінійні тенденції відповідають річній, тижневій та щоденній сезонності, а також ефектам відпусток. Найкраще це працює з часовими рядами, які мають сильний сезонний ефект та кілька сезонів історичних даних. Далі наведені прогнозні значення, розраховані за моделлю:
     - yhat - Прогнозоване значення часового ряду;

- yhat_lower - нижня межа інтервалу невизначеності;
- yhat_upper - верхня межа інтервалу невизначеності.

3) Візуалізація - ми можемо візуалізувати метричну поведінку часових рядів, створюючи графіки в Графані. Наприклад, ми можемо побудувати та порівняти фактичні та прогнозовані значення метрики за допомогою наступної інформаційної панелі, що зображена на рискунку 2.11:

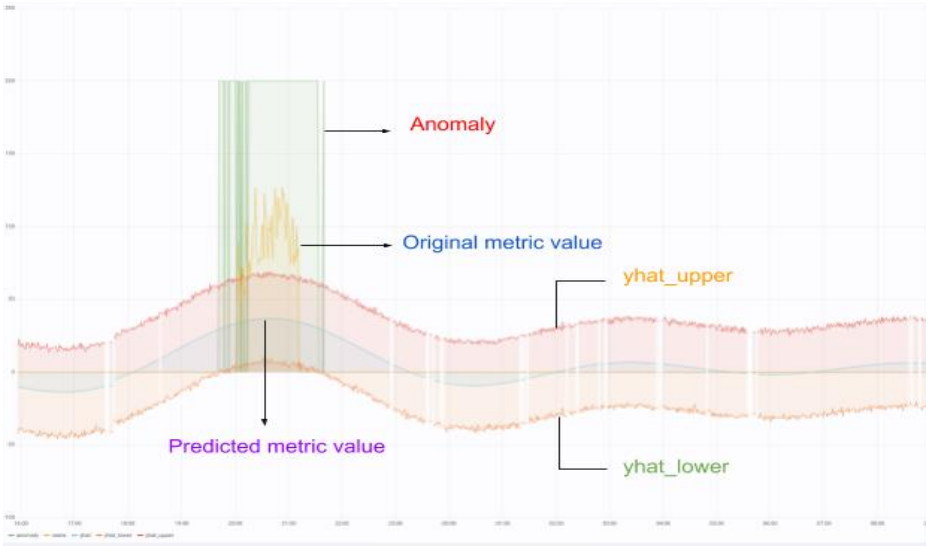

Рисунок. 2.11 - Приклад передбачення значень метрики, верхня та нижня межі для прогнозованого значення та виявленої аномалії.

4) Сповіщення: виявлені аномалії надсилаються як сповіщення за допомогою Prometheus AlertManager. Сповіщення можуть бути налаштовані на надсилання автоматичних повідомлень через чати Google та електронну пошту, або ж на будь-яке API повідомляючи про це відповідні команди. Для заданого часового проміжку метрики детектор аномалії Prometheus можна також запустити в "тестовому режимі", щоб перевірити, чи повідомили моделі машинного навчання про ці аномалії. Точність та продуктивність моделей потім можуть реєструватися як показники для MLFlow для порівняння результатів.

MLflow - це платформа з відкритим кодом для управління життєвим циклом ML, включаючи експерименти, відтворюваність. та розгортання. Наразі він пропонує три компоненти:

- відстеження Mlflow;
- проекти Mlflow;
- моделі Mlflow.

Тестовий режим корисний для відстеження модельних експериментів на локальній робочій станції, впорядкування коду в проектах для подальшого повторного використання та виведення найкращої моделі, яка буде розгорнута для виробництва.

2.4.1 Візуалізація інформації та спостережень за допомогою Grafana

Виявлення аномалії може працювати в режимі реального часу, наприклад, як передбачення невдачі процесу читання-запису бази даних.

Виявлення корисне тим, що кожного разу, коли прогнозується аномалія, буде помічено, що база даних справді не працює. Забезпечення виявлення аномалії може перевірятись встановленням сповіщення.

Приклад виявлення аномалії в Grafana зображено на рисунку 2.12:

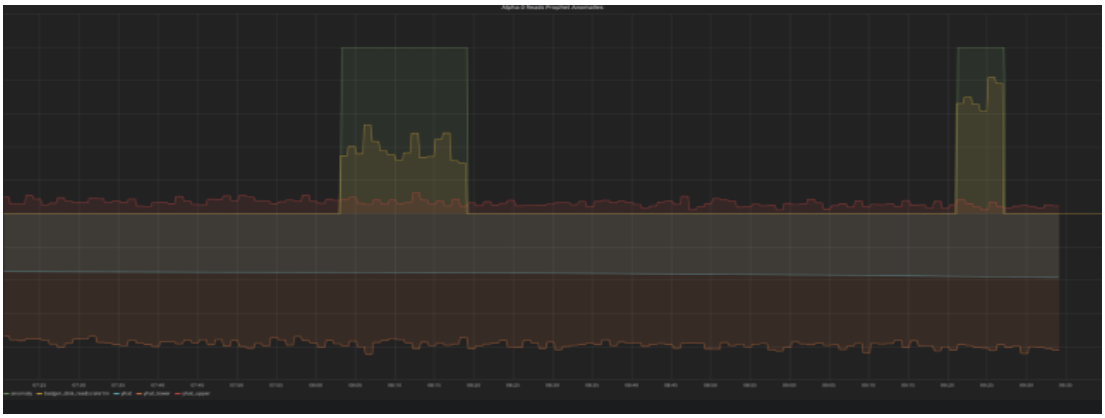

Рисунок. 2.12 - Приклад інформаційна панель Grafana: виявлення аномалії під час обслуговування кластера PSI

2.5 Висновок до розділу 2

На сьогоднішній день моніторинг є одним із найважливіших бізнес-процесів в житті IT-компаній. Зазвичай виділяють окремий департамент, який займається суто задачами моніторингу, такими як: вибір оптимального рішення, імплементацією, налаштуванням, підтримкою та реагуванням на збої в роботі IT-інфраструктури. Адже збої в системах можуть призвести до значних фінансових втрат.

Моніторинг IT-інфраструктури - це розгортання вбудованої бази знань для автоматичного діагностування проблем продуктивності та доступності всіх компонентів інфраструктури. Повний стек моніторингу IT-інфраструктури включає:

- моніторинг обладнання - фізичний стан системи;
- моніторинг ОС - працездатність та використання ресурсів;
- моніторинг мережі - споживання пропускної здатності та помилки;
- моніторинг додатків - продуктивність та доступність.

Оскільки IT-інфраструктури часто складається з декількох локацій розгортання компонентів (приватні, громадські та гібридні хмарні сховища) перед моніторингом стоїть завдання, якнайшвидче визначити і співвіднести проблеми, перш ніж вони вплинуть на кінцевих споживачів і в кінцевому підсумку на продуктивність організації.

В даному розділі були розглянуті основні принципи вибору систем моніторингу, опис популярних рішень, а також приведений приклад використання Prometheus та Grafana з використанням технологій штучного інтелекту, що в результаті дало ефективне рішення прогнозування проблеми нестачі фізичної пам’яті, а також виявлення проблем запису/читання бази даних.

РОЗДІЛ 3 УПРАВЛІННЯ КОНФІГУРАЦІЯМИ ЯК ЗАСІБ АВТОМАТИЗАЦІЇ ПРОЦЕСІВ

Як більш широка предметна область, управління конфігурацією (configuration management) відноситься до процесу систематичного внесення змін до системи таким чином, щоб вона підтримувала цілісність у часі. Незважаючи на те, що цей процес не виник у IT-індустрії, цей термін широко використовується для позначення управління конфігурацією сервера.

Автоматизація відіграє важливу роль в управлінні конфігурацією сервера. Це механізм використовується для того, щоб сервер досяг бажаного стану, який раніше визначався виконанням скриптів із використанням конкретної мови та функцій інструмента. Фактично, автоматизація є серцем управління конфігурацією для серверів, і тому, зазвичай, інструменти управління конфігурацією називають інструментами автоматизації або засобами автоматизації IT-інфраструктури.

Іншим поширеним терміном, який використовується для опису функцій автоматизації, реалізованих інструментами управління конфігурацією, є оркестрація сервера, оскільки інструменти, що виконують функцію оркестрації, як правило, здатні керувати безліччю серверів з центральної машини - контролера.

На ринку є ряд інструментів управління конфігурацією. Puppet, Chef та Ansible - найпопулярніший вибір. Хоча кожен інструмент має свої особливості та працює дещо по-різному, всі вони керуються однією і тією ж метою: переконатися, що стан системи відповідає стану, описаному сценаріями забезпечення.

3.1 Огляд процесу управління конфігураціями

Процес управління конфігурацією забезпечує, щоб вибрані компоненти IT-інфраструктури, системи або продукту (елементу конфігурації) були ідентифіковані, а зміни до них підтримувалися і контролювалися. Він надає модель конфігурації служб, активів та інфраструктури, записуючи взаємозв'язки між сервісними активами та елементами конфігурації. Він також забезпечує внесення змін у контрольоване середовище та експлуатаційне використання на основі офіційних схвалень.

Будь-який компонент, який потребує управління для надання IT-послуги, вважається частиною процесу управління конфігурацією.

Частина управління активами цього процесу керує активами сервісу протягом усього життєвого циклу, від придбання до вибуття. Цей процес також забезпечує повну інвентаризацію активів та пов'язаних з ними власників, відповідальних за їх контроль.

Мета управління конфігурацією - визначити та централізовано контролювати компоненти IT-інфраструктури, а також підтримувати точну інформацію про конфігурацію.

Ефективне управління конфігурацією сприяє більшій доступності системи, мінімізує виробничі проблеми та вирішує проблеми більш ефективно.

Управління конфігурацією включає п'ять основних видів діяльності. Процес управління конфігурацією охоплює всі ці дії та забезпечує ефективність відстеження та контролю ефективності активів. Основними видами діяльності в рамках управління конфігурацією є:

1) Планування управління конфігурацією - включає заходи, які дозволяють спланувати функцію, сферу та цілі менеджера служби для організації.
2) Ідентифікація конфігурації - включає заходи, які дозволяють ідентифікувати та маркувати всі існуючі IT-компоненти компанії.

Інформація, яка відстежується, включає ідентифікацію активів, відносини мережі активів та дані моделі чи версії.

3) Контроль конфігурації - включає в себе дії, які дозволяють гарантувати, що вся інформація щодо IT-компонентів буде оновлена та точна. Компоненти можна додавати, змінювати чи вилучати лише за допомогою контрольної документації, наприклад затвердженого запиту на зміну. Також головне управління даними включає в себе дії, які дозволяють узгодити головні контрольні дані, керовані в інших підрозділах IT-компанії.
4) Облік та звіт про стан конфігурації - включає заходи, які дозволяють надавати звіти про поточні та історичні дані, що стосуються кожного IT-компонента протягом його життєвого циклу. Облік стану вносить зміни до компонентів, які можна відслідковувати.
5) Перевірка та аудит конфігурації - включає заходи, які дозволяють перевірити фізичне існування IT-компонентів та забезпечити їх правильну ідентифікацію.

Загальний огляд процесів управління конфігурацією та робочих процесів зображено намалюнку 3.1:

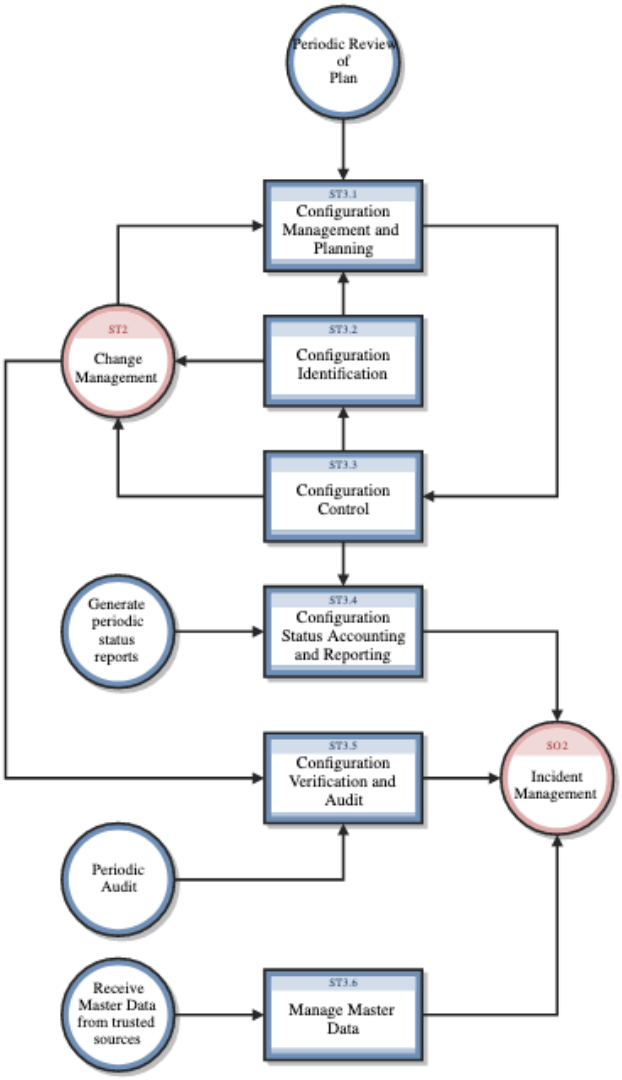

Рисунок. 3.1 - Блок-схема роботи процесу управління конфігурацій

3.2 Переваги керування конфігурацією для серверів

Хоча використання засобів управління конфігурацією зазвичай вимагає більше зусиль та початкового планування, ніж управління вручну, проте, все, крім найпростішої серверної інфраструктури, буде покращено за рахунок переваг, які вона надає. Перерахуємо декілька:

1) швидке забезпечення нових серверів;

2) швидке відновлення стану інфраструктури після критичних подій;
3) легке керування групами вузлів;
4) контроль версій для середовища сервера;
5) реплікація середовищ.

Кожен раз, коли потрібно розгорнути новий сервер, інструмент управління конфігурацією може автоматизувати більшу частину, якщо не весь процес підтримки існуючих середовищ. Автоматизація робить розгортання та внесення змін набагато швидше та ефективніше, оскільки дозволяє виконувати виснажливі завдання швидше та точніше, ніж могла б будь-яка людина. Навіть за умови належної та ретельної документації, наприклад, вручну розгортання веб-сервера може зайняти години порівняно з кількома хвилинами з управлінням/автоматизацією конфігурації.

Швидке внесення змін забезпечує ще одну перевагу: швидке відновлення працездатного стану системи від наслідків критичних подій. Коли сервер переходить у режим офлайн через невідомі обставини, для належного аудиту системи та з’ясування того, що сталося, може знадобитися кілька годин. У подібних сценаріях розгортання сервера заміни, як правило, є найбезпечнішим способом повернення служб в продуктове середовище, тоді як детальний огляд може проводитися на постраждалому сервері без впливу на кінцевого споживача. Засоби управління конфігурацією та автоматизація дозволяє це зробити швидко та надійно.

На перший погляд, ручне адміністрування системи може здатися простим способом розгортання та швидкого виправлення серверів, але це часто пов'язано із зайвим витрачанням ресурсів, необхідність в яких часом є критичною. З часом може бути надзвичайно важко дізнатися, що саме встановлено на сервері та які зміни були внесені, коли процес не автоматизований. Виправлення вручну, налаштування конфігурації та оновлення програмного забезпечення можуть перетворити сервери в унікальні середовища, важкі в управлінні та ще складніші для копіювання. Використовуючи інструмент управління конфігурацією,

процедура, необхідна для створення нового сервера або оновлення існуючого, буде задокументована в сценаріях внесення змін.

Після того, як налаштування сервера переведено в набір сценаріїв, з'явиться можливість застосувати до серверного середовища багато інструментів та робочих процесів, які зазвичай використовуються для корегування вихідного коду програмного забезпечення.

Засоби контролю версій, такі як Git, можуть використовуватися для відстеження змін, внесених до забезпечення, та для підтримки окремих гілок застарілих версій скриптів. Також існує можливість використовувати систему контролю версій для реалізації політики перегляду коду для сценаріїв, де будь-які зміни повинні бути подані як запит на додавання версії та затверджені керівником проекту перед тим, як зафіксуватись. Подібна практика додасть додаткової послідовності налаштуванням інфраструктури.

Управління конфігурацією робить реальним процес копіювання середовища з точно таким же програмним забезпеченням та конфігураціями. Це дає змогу ефективно будувати багатокомпонентні середовища з серверами виробництва, розробки та тестування. Також з’являється можливість використовувати локальні віртуальні машини для розробки, побудовані однаковими сценаріями розгортання. Така практика дозволить мінімізувати проблеми, спричинені розбіжностями середовищ розробки, які часто виникають під час розгортання програм для виробництва або спільного використання між колегами з різними налаштуваннями машин (різною операційною системою, версіями програмного забезпечення та / або конфігураціями).

3.3 Особливості інструментів управління конфігурацією

Навіть незважаючи на те, що кожен інструмент УК має свої терміни, філософію та вимоги до середовища, вони, як правило, мають багато схожих характеристик і подібних понять.

Більшість інструментів управління конфігурацією використовують модель контролера (головного вузла) і агента (підконтрольного вузла). По суті, контролер керує конфігурацією підконтрольних вузлів на основі низки інструкцій або завдань, визначених у сценаріях забезпечення.

Нижче перерахуємо найпоширеніші функції, присутні в більшості інструментів УК для серверів:

1) фреймворк автоматизації;
2) ідемпотентна поведінка;
3) інформацію про систему;
4) шаблонна система;
5) розширюваність.

Кожен інструмент УК має певний синтаксис декларування завдань та набір функцій, які можна використовувати для написання сценаріїв забезпечення. Більшість інструментів підтримує функції, які роблять їхню мову подібною до звичайних мов програмування, але спрощеним чином. Змінні, цикли та умовні параметри є загальними ознаками, що сприяють створенню більш універсальних сценаріїв забезпечення.

Інструменти управління конфігурацією відстежують стан ресурсів, щоб уникнути повторення завдань, які виконувалися раніше. Якщо пакет уже встановлений, інструмент не буде намагатися встановити його знову. Мета полягає в тому, що після кожного запуску забезпечення система досягає (або зберігає) бажаний стан, навіть якщо виконати його декілька разів. Це те, що характеризує ці інструменти як такі, що мають самовпевнену поведінку. Однак ця поведінка не обов'язково виконується у всіх випадках.

Засоби УК зазвичай надають детальну інформацію про систему, що підтримується. Ці дані доступні через глобальні змінні, відомі як факти. Вони включають такі речі, як мережеві інтерфейси, IP-адреси, інформацію про версію операційної системи та дистрибутив (ядро). Кожен інструмент надасть різний набір фактів. Їх можна використовувати, щоб зробити сценарії та шаблони забезпечення більш адаптивними для декількох систем.

Більшість інструментів УК надають вбудовану систему шаблонів, яку можна використовувати для полегшення процесу налаштування файлів та служб конфігурації. Шаблони зазвичай підтримують змінні, цикли та умовні параметри, які можна використовувати для досягнення максимальної універсальності. Наприклад, використання шаблонів для легкого налаштування нового віртуального хоста з веб-сервером Apache, використовуючи при цьому повторне виконання одного і того ж шаблона для декількох налаштувань сервера. Замість того, щоб мати тільки жорстко закодовані статичні значення, шаблон повинен містити змінні значень, які можуть змінюватися від хоста до хосту, таких як NameServer та DocumentRoot.

Незважаючи на те, що сценарії можуть бути дуже спеціалізованими для потреб певного сервера, є багато випадків, коли можливі подібні налаштування сервера або частини налаштувань, які можна поділити між декількома серверами.

Сторонні модулі та плагіни можливо легко знайти, спеціально для таких загальних налаштувань сервера, як, наприклад, інсталяція веб-сервера, чи навіть бази даних. Інструменти УК мають, як правило, велику за обсягом спільноту, в якій користувачів заохочують ділитися своїми сценаріями. Використання готових сценаріїв, наданих іншими користувачами, може заощадити багато часу, а також служить відмінним способом дізнатися, як інші користувачі вирішували поширені проблеми, використовуючи потрібний інструмент.

3.4 Вибір інструмента управління конфігурації

На ринку існує багато інструментів УК, кожен з яких має різний набір функцій та різний рівень складності. Популярні варіанти включають Chef, Ansible та Puppet. Перше завдання - обрати інструмент, який найкраще відповідає висунутим вимогам та наявним потребам. Перед вибором слід врахувати кілька речей:

1) складність інфраструктури;
2) складність інструменту;
3) ціна затрачених ресурсів;
4) розширений інструментар;
5) спільнота та підтримка.

Більшість інструментів управління конфігурацією потребують мінімальної ієрархії, що складається з машини контролера та вузла, яким він керує. Наприклад, Puppet вимагає встановлення програми агента на кожен вузол, а на машині контролера необхідно встановити головну програму. З іншого боку, Ansible має децентралізовану структуру, яка не потребує встановлення додаткового програмного забезпечення на вузлах, але виконання завдань сценарію відбувається через протокол віддаленного керування SSH. Для менших проектів спрощена інфраструктура може здатися більш придатною, однак важливо враховувати такі аспекти, як масштабованість та безпека, які інструмент може не підтримувати.

Деякі інструменти можуть мати більше компонентів та рухомих деталей, що може збільшити складність інфраструктури, впливаючи на потребу додаткового навчання та, можливо, збільшуючи загальну вартість впровадження.

Як було згадано раніше, інструменти УК надають користувацький синтаксис, іноді використовуючи конкретну доменну мову (DSL), і набір функцій, що складають їх основу для автоматизації. Як і у звичайних мовах програмування, деякі інструменти вимагають освоїти додаткові знання. Вимоги

до інфраструктури також можуть вплинути на складність інструменту та на те, як швидко з’явиться можливість побачити рентабельність інвестицій ресурсів на імплементацію нового рішення.

Більшість інструментів УК пропонують безкоштовні або відкриті версії з платними підписками з розширенними функціями та послугами. Деякі інструменти матимуть більше обмежень, ніж інші, тому залежно від конкретних потреб та того, яким чином зростає інфраструктура, можливо, доведеться платити за ці послуги. Також необхідно розглядати навчання як потенційну додаткову вартість не лише у грошовому еквіваленті, а й щодо часу, який знадобиться для того, щоб команда відшкодувала затрачений час на дослідження та імплементацію обраного інструменту.

Як зазначалося раніше, більшість інструментів пропонують платні послуги, які можуть включати підтримку та розширений інструментарій. Важливо проаналізувати конкретні потреби, розмір інфраструктури та необхідність використання цих послуг. Наприклад, панелі управління - це звичайна послуга, що пропонується в більшості інструментів, і вони можуть значно полегшити процес управління та моніторингу всіх серверів з центрального вузла. Навіть якщо такі послуги ще не потрібні, необхідно розглянути варіанти можливої майбутньої необхідності.

Сильна та привітна спільнота може бути надзвичайно сприятливою для підтримки та документації, оскільки користувачі, як правило, раді поділитися своїми знаннями та напрацюваннями (модулями, плагінами та готовими сценаріями) з іншими користувачами, якщо це, звичайно, не суперечить правилам компаній. Це може бути корисним для прискорення процесу навчання та уникнення зайвих витрат на платну підтримку.

3.5 Огляд та порівняння популярних засобів УК

3.5.1 Ansible

Ansible - це програмний інструмент, який забезпечує просту, але потужну автоматизацію для підтримки середовищ розробки. Він в першу чергу призначений для IT-фахівців, які використовують його для розгортання додатків, оновлень на робочих станціях та серверах, надання хмарних послуг, управління конфігурацією, оркестрування внутрішньої служби та майже все, що системний адміністратор робить щотижня або щодня. Ansible не залежить від програмного забезпечення агента (окрім вимоги наявності встановленого пакету Python, який встановлений за замовчуванням в більшості актульних дистрибутивах ОС) і не має додаткової інфраструктури безпеки, тому його легко розгортати.

Оскільки Ansible стосується автоматизації, для виконання кожного кроку потрібні інструкції. З урахуванням всього, що записується декларативною мовою у простій формі скрипту формату YAML, досить легко керувати версіями компонентів та підтримувати ідемпотентність системи. Практичний результат цього - головний внесок у рух IaaC: ідея про те, що підтримка серверної та клієнтської інфраструктури може трактуватись так само, як і розробка програмного забезпечення: зберігання версій перевірених і затвердженні рішення сценаріїв в системах контролю версій, що здатні керувати організацією незалежно від змін персоналу.

Хоча Ansible може бути в авангарді автоматизації, адміністрування систем та методології DevOps, він також корисний для щоденних потреб користувачів. Ansible дозволяє налаштувати не один комп'ютер, а потенційно цілу групу комп’ютерів одночасно, а для його використання не потрібні навички програмування. Інструкції, написані для Ansible, легко читаються як новачками, так і досвідченими розробниками.

В Ansible існує дві категорії комп'ютерів: вузол управління та керовані вузли. Вузол управління - це комп'ютер, на якому встановлений та працює Ansible. Повинен бути принаймні один вузол управління, хоча також може існувати і резервний вузол управління для збереження доступу до інфраструктури у разі виникнення критичних ситуацій. Керований вузол - це будь-який пристрій, яким керує вузол управління.

Ansible працює за допомогою підключення до вузлів (клієнтів, серверів чи будь-чого, що ви налаштовуєте) в мережі, а потім надсилає до цього вузла невелику програму під назвою модуль Ansible. Ansible виконує ці модулі через SSH та видаляє їх після закінчення. Єдина вимога до цієї взаємодії полягає в тому, щоб ваш вузол управління Ansible мав доступ до керованих вузлів. SSH-ключі - найпоширеніший спосіб надання доступу, але підтримуються й інші форми аутентифікації.

Термін модулі звучить складно, але більшу частину складності обробляє Ansible, а не користувач. Модуль Ansible пишеться як модель бажаного стану системи, тобто кожен модуль визначає, що має бути застосовано для будь-якого керованого вузла. Наприклад, якщо системний адміністратор вирішив, що на всіх робочих станціях організації має бути встановлена версія LibreOffice версії X.Z, то перш за все слід встановити, чи кожен вузол має LibreOffice X.Z. Якщо Ansible знайде керований вузол із встановленим LibreOffice X.Y, він ідентифікує тип дистрибутиву та версію операційної системи, а після цього запустить необхідну процедуру для оновлення LibreOffice до версії X.Z. Таким чином, кожну робочу станцію в організації можна оновлювати протягом ночі за допомогою програмного забезпечення, що підтримується IT-відділом.

Однак підтримка інфраструктури - це не просто перевірка версій програмного забезпечення. Коли річ йде про використання Ansible, то зазвичай мають на увазі використання модулів Ansible, оскільки це частини Ansible, які виконують конкретні завдання. При необхідності щось автоматизувати на кількох комп’ютерах, варто переглянути існуючі модулі Ansible, щоб знайти той, який вирішує завдання, яке потрібно виконати, а потім встановіти його в Ansible

для подальшої змоги налаштування та виклику цього модуля. Також існує можливість написати власні спеціальні модулі для виконання спеціалізованих завдань. Окрім того, завжди є змога запропонувати самостійно розроблений модуль до переліку офіційно включених модулів, щоб інші користувачі мали могу вільно його використовувати.

Хоча модулі є засобами для виконання завдань, спосіб їх використання здійснюється через так звану “книгу ігор” (playbook) Ansible. Playbook - це файл конфігурації, написаний у форматі YAML, який дає інструкції щодо того, що потрібно зробити, щоб привести керований вузол у потрібний стан. Книги ігор повинні бути простими, читабельними для людей та самостійними сценаріями. Вони універсальні для інших систем, це означає, що її можна запускати в системі в будь-який час без негативноговпливу. Якщо playbook запускається в системі, яка вже належним чином налаштована і перебуває у бажаному стані, то ця система все одно має бути належним чином налаштована після виконання сценарію.

Сценарій (playbook) може бути дуже простим, наприклад, таким, який встановлює як привілейований користувач (root) HTTP-сервер Apache на будь-якому вузлі групи веб-серверів IT-відділу. Але в той же час може бути дуже складними, з умовами та змінними. Однак, оскільки більшість реальної роботи виконується модулями Ansible, “плейбуки” зазвичай залишаються короткими, читабельними та зрозумілими, хоча вони можуть упорядкувати цілі групи керованих вузлів.

3.5.2 Puppet

Puppet є одним з найбільш використовуваних інструментів управління конфігурацією. Як і Ansible - це система внесення необхідних змін до системи з метою забезпечення її цілісності протягом тривалого часу. Він забезпечує

фіксування та аудит стану системи. Як засіб УК дозволяє подолати наступні перешкоди:

1) в разі невдалого виконання сценарію, здатен повернути систему до попереднього стану;
2) здатен гнучко реагувати на зміни у вимогах;
3) якщо вимоги змінилися з моменту останнього впровадження, здатен успішно виконати потрібні зміни.

Puppet надає можливість визначати, яке програмне забезпечення та конфігурацію вимагає система, а потім підтримувати вказаний стан після початкової установки.

Для визначення параметрів конфігурації для конкретного середовища або інфраструктури використовується декларативна мова (DSL), подібна до Ruby. Puppet виявляє інформацію про систему за допомогою утиліти під назвою Facter, яка встановлюється під час встановлення програмного пакету Puppet.

Puppet працює за допомогою режиму витягування, коли агенти запитують головного майстра через фіксовані проміжки часу для отримання конфігурацій, характерних для конкретних вузлів. У такій інфраструктурі керовані вузли запускають додаток Puppet agent, як правило, як фоновий сервіс.

3.5.2.1 Компоненти та принципи використання Puppet

Нижче приведено основні принципи архітектури Puppet:

- Puppet Master: головний сервер, який керує конфігурацією на керованих вузлах;
- вузол Puppet агента: вузол, яким керує Puppet Master;
- маніфест: файл, що містить набір інструкцій, які потрібно виконати;

- ресурс: частина коду, яка оголошує елемент системи та те, як слід змінити її стан. Наприклад, для встановлення пакету потрібно визначити ресурс пакета та переконатися, що його стан є "installed";
- модуль: набір маніфестів та інших пов’язаних файлів, організованих заздалегідь визначеним способом для полегшення обміну та повторного використання частин передбаченого сценарію;
- клас: як і у звичайних мовах програмування, класи використовуються для того, щоб краще організувати розгортання та полегшити повторне використання частин коду;
- факти: глобальні змінні, що містять інформацію про систему, наприклад, мережеві інтерфейси та операційну систему;
- сервіси: використовуються для зміни стану сервісу , наприклад перезавантаження або зупинення сервісу.

3.5.2.2 Корисні випадки використання Puppet

Нижче наведено перелік прикладів використання коли Puppet є найбільш доречним засобом УК:

- коли необхідно надавати чіткі конфігурації для кожного серверу;
- коли необхідні постійні перевірки, щоб підтвердити, чи потрібна конфігурація задіяна та чи не змінилась;
- коли необхідно скоротити витрати та зусилля, коли доводиться змінювати незначний код у сотнях систем;
- коли необхідно надавати чіткий та перевірений механізм контролю змін;
- коли необхідно проводити оновлення програмних пакетів всієї інфраструктури та ідемпотентність системи є обов’язковою умовою;
- коли необхідно тестувати потрібні зміни в тестових середовищах за допомогою автоматизації. Можливості автоматизації за допомогою Puppet

великі, і саме тому більшість інженерів DevOps методології обирають саме цей інструмент.

3.5.3 Chef

Chef – як і попередні розлянуті інструменти УК, це інструмент управління конфігурацією, який керує інфраструктурою, відповідно до написаного коду, за домогою якого можна легко автоматизувати тестування та розгортання необхідних програмних компонентів, або ж внесення змін до конфігурації ОС. Як і Puppet, Chef має архітектуру мастер-агент і він підтримує декілька платформ, таких як Windows, Ubuntu, Centos і Solaris і т.д.

Chef здатен виконати автоматизацію конфігурації, розгортання та управління додатками по всій мережі. Chef - чудовий інструмент для прискорення доставки програмного забезпечення, швидкість розробки програмного забезпечення залежить від того, наскільки швидко програмне забезпечення здатне змінитися у відповідь на нові вимоги або умови.

Chef в основному складається з трьох компонентів, Chef-сервера, робочих станцій та вузлів. Як і у випадку з Puppet, сервер Chef є центром усіх операцій, де задаються та зберігаються зміни. Робоча станція - це місце, створюються або змінюються конфігурації. Вузли - це машини, якою керує Chef.

Користувач має змогу взаємодіяти з Chef-сервером з віддаленої робочої станції, таким чином немає гострої необхідності розгортати Chef локально, як у випадку з Ansible. Інструменти командного рядка Knife використовуються для взаємодії з Chef-сервером. Chef вузол - це сервер, яким керує Chef, і кожен вузол налаштовується встановленим на ньому Chef-клієнт (по аналогії з Puppet-агентом). У свою чергу, Chef-сервер зберігає всі дійсні конфігурації. Це гарантує, що всі елементи знаходяться в потрібному місці та працюють як слід.

3.5.3.1 Перваги Chef

Наведемо основні переваги застосування Chef:

1) прискорення доставки програмного забезпечення;
2) підвищена стійкість служби;
3) управління ризиками;
4) адаптація до хмарної інфраструктури;
5) керування центрами даних та хмарним середовищем;
6) упорядковані IT-операції та робочий процес.

Коли інфраструктура автоматизована, всі програмні вимоги, такі як тестування, створення нових середовищ для розгортання програмного забезпечення тощо, стають значно швидшими.

Зробивши інфраструктуру автоматизованою, з'являється можливість відстежувати помилки до їх виникнення, також це значно пришвидшує відновлення працездатності системи після виникнення критичних ситуацій.

Як зазначалось раніше Chef може працювати на різних платформах та дозволяє керувати всіма серверними ресурсами, де б вони не розташовувались.

3.5.3.2 Огляд компонентів Chef

Chef містить наступні компоненти:

- Chef-сервер: центральний сервер, який зберігає інформацію та керує забезпеченням вузлів;
- Chef-вузол: індивідуальний сервер, яким керує Chef-сервер;
- робоча станція: контролер, на якому створюються сценарії розгортання та завантажуються до Chef-сервера;

- рецепт: файл, що містить набір інструкцій (ресурсів), які потрібно виконати. Рецепт повинен міститись у кулінарній книзі;
- ресурс: частина коду, яка декларує елемент системи та яку дію слід до нього застосувати. Наприклад, для встановлення пакету декларується ресурс пакета разом із дією “install”;
- кулінарна книга: збірка рецептів та інших пов’язаних файлів, організованих заздалегідь визначеним способом для полегшення обміну та повторного використання частин сценарію;
- атрибути: відомості про певний вузол. Атрибути можуть бути автоматичними, а також можуть визначатись всередині рецептів;
- автоматичні атрибути: глобальні змінні, що містять інформацію про систему, наприклад мережеві інтерфейси та операційну систему (відомі як факти в інших інструментах УК). Ці автоматичні атрибути збираються інструментом під назвою “Ohai”;
- сервіси: використовуються для зміни стану певного сервісу, наприклад перезавантаження або зупинення.

Варто зазначити, що для створення кулінарних книг використовується мова Ruby, а для конкретних ресурсів використовуються мова DSL (Domain specific language).

3.5.4 Порівняння популярних інструментів

Наведена нижче таблиця 3.1 описує основні відмінності між трьома розглянутими інструментами УК: Ansible, Puppet та Chef.

Таблиця 3.1 – Порівняння інструментів управління конфігурації

	Ansible	Puppet	Chef
Мова написання скриптів (сценарію)	YAML	Спеціальний DSL на основі Ruby	Ruby
Необхідні компоненти інфраструктури	Сервер контролера застосовує конфігурацію на вузлах через SSH	Puppet-севрер синхронізує конфігурацію на Puppet-вузлах за допомогою Puppet-агентів	Робочі станції Chef підштовхують конфігурацію до Chef-серверу, звідки Chef-вузли буде оновлено
Потрібно спеціалізоване програмне забезпечення для вузлів	Ні (за винятком Python, який у більшості дистрибутивах ОС встановлений за замовчуванням).	Так.	Так.
Забезпечує централізований пункт контролю	Ні. Будь-який комп'ютер може бути контролером	Так, через Puppet-сервер	Так, через Chef-свервер
Термінологія сценаріїв	Playbook / Roles	Manifests / Modules	Recipes / Cookbooks
Виконання сценаріїв	Послідовне	Паралельне	Послідовне

3.6 ВИСНОВОК ДО РОЗДІЛУ 3

Мета управління конфігурацією - визначити та централізовано контролювати компоненти IT-інфраструктури, а також підтримувати точну інформацію про її конфігурацію (перелік встановленого програмного забезбечення з відповідними параметрами, а також особливості налаштування ОС кожного серверу). В той же час ефективне використання засобів управління конфігурацією сприяє більшій доступності систем, мінімізує виробничі проблеми та вирішує проблеми більш ефективно.

В цьому розділі було розглянуто три популярних засобів УК: Ansible, Puppet та Chef. Кожен з розглянутих засобів є потужним інструментом, використання якого значно економить ресурси при обслуговуванні IT-інфраструктури, що включає в себе підтримку та розгортання різних середовищ розробки.

РОЗДІЛ 4 РОЗРОБКА СЦЕНАРІЮ АВТОМАТИЗОВАНОГО РОЗГОРТАННЯ СИСТЕМИ МОНІТОРИНГУ

В цьому розділі описаний процес автоматизованого розгортання моніторингу на базі Prometheus та Grafana із застосуванням експертних систем за допомогою інструменту УК – Ansible. Основною вимогою до середовища розгортання є сервери на базі ОС сімейства Linux. Додатково буде розглянуто процес виявлення аномалії за допомогою мови запитів Prometheus.

Перш за все ми зацікавлені в тому, щоб використовувати виключно безкоштовні рішення задля цілей автоматизації розгортання та побудови системи моніторингу.

Як було описано в розділі 3, інструмент Ansble не вимагає встановлення додаткових ресурсів та є повністю безкоштовним рішенням, саме тому його було обрано як засіб автоматичного розгортання системи моніторингу.

Поєднання компонентів Prometheus та Grafana надає багато можливостей, а також таке рішення не вимагає фінансових витрат.

4.1 Огляд структури файлів та параметрів сценарія розгортання Ansible

Як вже було зазначено раніше в розділі 3 використовуючи Ansible сценарії записуються в спеціальний файл формату YAML, який називається playbook.

Хоча і сценарій розгортання можна записати як один playbook, всередині якого містяться зміні хостів та перелік завдань, проте це не є гарною практикою. Подібний підхід є зовсім не гнучким, а подальше використання таких сценаріїв при зміні вимог може бути надзвичайно незручним та призвести до неочікуваного виконання через людський фактор.

Задля можливості універсального застосування розробленого сценарію розгортання кожного окремого компонента системи моніторингу було винесено як окрему роль, а змінні для зручності розбиті на групи. Таких підхід дозволяє гнучко виконувати завдання в залежності від конкретної потреби та в різних ситуаціях.

Для Ansible є надзвичайно критичною структура директорій та файлів, які програма шукає по спіралі, починаючи з поточного каталогу:

- ansible.cfg: головний конфігураційний файл Ansible, що містить в собі відомості про розташування конфігураційних директорій, таких як директорія з файлами кешування даних про підключення, директорія з логами діяльності Ansible та директорія з файлами збереження повторного запуску у разі невдалого застосування, а також файл inventory, що містить відомості про параметри підключення до керованих вузлів;
- inventory: як було зазначено раніше, це файл що містить важливі параметри підключення до керованих вузлів. Особливістю є те, що можливий розподіл вузлів на групи. Наприклад, якщо ми маємо декілька веб-серверів, ми можемо об’єднати їх в одну групу, що в подальшому дозволить застосувати перелік завдань по оновленню вузлів вуб-серверів як до групи, замість того, щоб перелічувати всі вузли окремо. Також в inventory файлі є можливість задання змінних для групи вузлів або до кожного вузла окремо;
- group_vars: директорія з файлами що містять перелік змінних, які застосовуються до групи вузлів. Це зручно у випадку, коли, наприклад, потрібно проконтролювати встановлення однієї і тої самої версії програмного компоненту на декілька вузлів одночасно.;
- host_vars: директорія з файлами, що містять змінні для кожного вузла окремо. Це потрібно, коли є необхідність вказати специфічні налаштування кожного вузла, наприклад, параметри налаштування мережі;

– roles: директорія, що забезпечує організацію різних, але пов’язаних між собою задач, а також розміщення зв’язаних з цими завданнями даних в одному місці. В нашому випадку будуть міститися наступні ролі: розгортання експортерів, Prometheus та Grafana.

Ролі мають свою структуру підкаталогів:

– files: містить файли, які можуть бути скопійовані на підконтрольні вузли, наприклад, скрипти, які потрібно виконати на віддаленому сервері, або вихідний код програмного забеспечення для подальшого його компілювання;
– handlers: обробники, які будуть запущені в разі успішного виконання завдань;
– meta: опис залежностей, тобто ролей, які мають бути обробленими та мета-даних, таких як автор, опис продукту та інше;
– templates: шаблони файлів зі змінними, задекларованими шаблонізатором Jinja2;
– tasks: всі задачі, що були раніше описані в файлі-сценарії. Варто зазначити, що головним файлом є main.yml, що повинен містити в собі всі інші сценарії, що мають виконатись;
– vars: змінні для шаблонів.

В додатку А наведено лістинг файлів, необхідних для розгортання системи моніторингу. Розглянемо основні модулі та параметри, що використовувались:

– модуль “template” копіює на керований хост відповідний шаблон, підставляючи вказані значення змінних; використовується для автоматичного створення потрібних файлів конфігурацій. Має наступні аргументи (параметри): “src” – назва шаблону у відповідній директорії ролі, “dest” – місце розташування на віддаленому сервері та параметри власників і прав (owner, group, mode);

- модуль “file” копіює на керований хост відповідні файли або директорії. Має наступні параметри: “path” – шлях куди має бути скопійовий файл або директорії, “state” – вказівка про те, що це файл або директорія, та параметри власників і прав;
- модуль “unarchive” виконує розархівацію на локальній або віддаленій машині. Має наступні параметри: “src” – розташування архіву на віддаленому вузлі, що має бути розпакованим, “dest” – шлях, за яким буде знаходитись вміст архіву, “remote_src” – вказівник знаходження архіву на віддаленому хості, “extra_opts” – додаткові параметри, що мають бути передані в масив;
- модуль “notify” викликає обробника (handlers).

4.2 Архітектура і конфігурація системи моніторингу

Нагадаємо, що експертна система має складатись з наступних компонентів:

1) база даних (не обов’язково);
2) база знань;
3) інтерпретатор;
4) інтерфейс користувача.

Натомість, побудована нами система моніторингу включає в себе такі компоненти, як:

1. Prometheus exporter – база даних. Роль експортера доволі проста, його ціль зібрати дані у вигляді метрик на цільовому вузлі та, підготувавши їх для Prometheus, очікувати на запит.
2. Prometheus - база знань та інтерпретатор. Основним компонентом нашої моніторингової системи є TSDB Prometheus. Нагадаємо, що людина-експерт має підготувати спеціальний файл, що містить перелік всіх

цільових експортерів, вказавши IP-аресу вузла та порт, який прослуховується відповідним експортером. В свою чергу Prometheus з вказаною періодичністю відсилає http(s) запит всім цільовим екпортерам на зібрані метрики (дані), які підготували експортери на цільовому вузлі. Отримавши відповідь, записує отримані дані в базу даних. Особливістю є те, що використовуючи мову запитів Prometheus людина-експерт має змогу обробляти та маніпулювати даними. Наприклад, вказати поріг сповіщення про навантаження на процесор чи заповнення простору на диску. Подібні правила записуються в спеціальні файли, місцерозташування яких необхідно вказати в головному конфігураційному файлі.

3. Grafana – інтерфейс користувача. Вказавши необхідне джерело даних (в нашому випадку це наш TSDB Prometheus), Grafana має доступ до всіх даних, що там зберігаються. Людина-експерт має або імпортувати готові, або створити самостійно інформаційні панелі, які в подальшому будуть слугувати інтерфейсом користувача та візуалізувати стан компонентів, що перебувають під моніторингом. Grafana має змогу сповіщати користувача про настання певних подій, які були попередньо вказані людиною-експертом. Наприклад, відсоток HTTP-відповідей з кодом 4xx або 5xx перевищує вказану норму, або відсоток навантаження процесора на веб-сервері занадто великий і так далі.
4. Prometheus AlertManager - вирішувач. Основна роль цього компонента сповіщати про події. Але AlertManager здатний не просто показати сповіщення про подію, а надсилати відповідні запити на вказані API-сервери, що дає змогу встановити та автоматизувати гнучкі рамки реагування на події, котрі були виявлені. Наприклад, у випадку коли один з веб-серверів не відповідає система автоматично ініціалізує ще один та додає його в балансувальник навантаження. Подібна схема широко використовується в AWS EC2 Autoscale Group.

На рисунку 4.1 зображено схему взаємодії компонентів побудованої нами системи моніторингу:

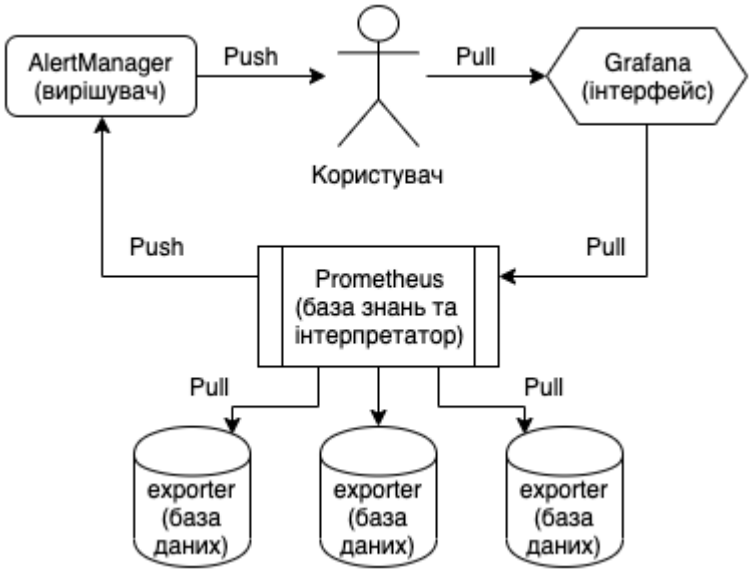

Рисунок. 4.1 – схема взаємодії компонентів побудованої системи моніторингу

Конфігурація екпортера Prometheus не вимагає значних зусиль, але в той же час є надзвичайно важливою, адже від цього залежить які саме дані будуть надсилась в Prometheus – базу знань. Зазвичай, експортер представляється як бінарний файл написаний на мові програмування Go. Основні вимоги до експортеру це те, що він має бути запущений у фоновому режимі, в деяких випадках з певним набором вхідних параметрів. Також експортер може працювати всередині Docker-контейнеру, якщо того вимагає середовище розробки, де буде розгорнуто сам експортер. Для отримання даних про використання ресурсів хосту (споживання ЦПУ, пам’яті, фізичного дискового простору), а також відомостей про його стан та доступність ми використали офіційний експортер від Prometheus, а саме “Node exporter”. Розгортання відбулося за допомогою “systemd”, де експортер був запущений у фоновому режимі як сервіс. Лістинг параметрів сервісу наведений в додатку А.

Конфігурація Promeheus дещо складніше, але так само не вимагає значних зусиль. Після розгортання потрібних програмних компонентів основна конфігурація зберігається в наступних файлах:

– prometheus.yml містить головні параметри конфігурації, такі як “scrape_interval” (інтервал зчитування метрик з експортерів) , “rule_files” (розташування правил для AlertManager), “ targets” (перелік розташування експортерів у форматі ip:port).
– alertmanager.yml містить параметри конфігурації AlertManager, наприклад параметри підключення до поштового серверу або інших засобів сповіщення, а також параметри надсилання сповіщень.
– rules містить перелік правил, по яким відбувається сповіщення у разі виявлення події.

Конфігурація Grafana дещо відрізняється від конфігурації інших компонентів системи монітингу. Це зумовлено тим, що зі сторони серверу необхідно проінсталювати пакет, а всі інші налаштування доступу та інформаційних панелей здійснюються через веб-інтерфейс.

На рисунках 4.2, 4.3, 4.4, 4.5 та 4.6 зображена поетапна конфігурація Grafana, в тому числі підключення Prometheus та встановлення інформаційної панелі навантаження серверу:

Рисунок. 4.2 – Сторінка авторизації до веб-інтерфейсу

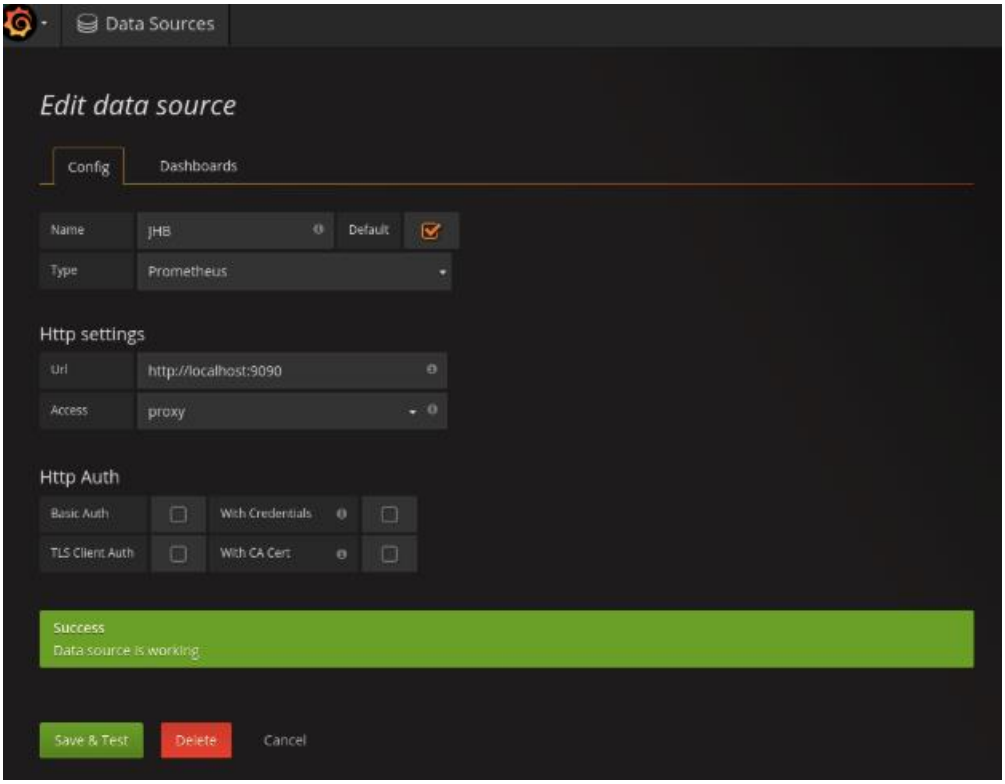

Рисунок. 4.3 – Сторінка редагування джерела метрик

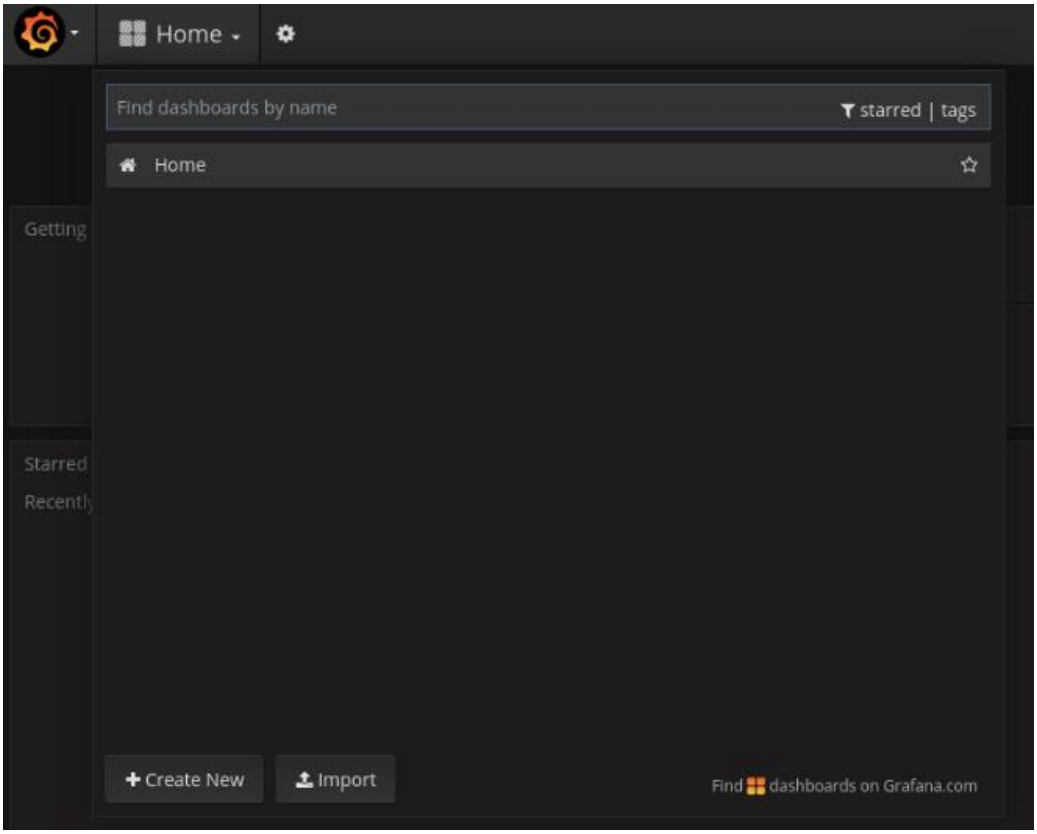

Рисунок. 4.4 – Сторінка пошуку необхідних інформаційних паналей

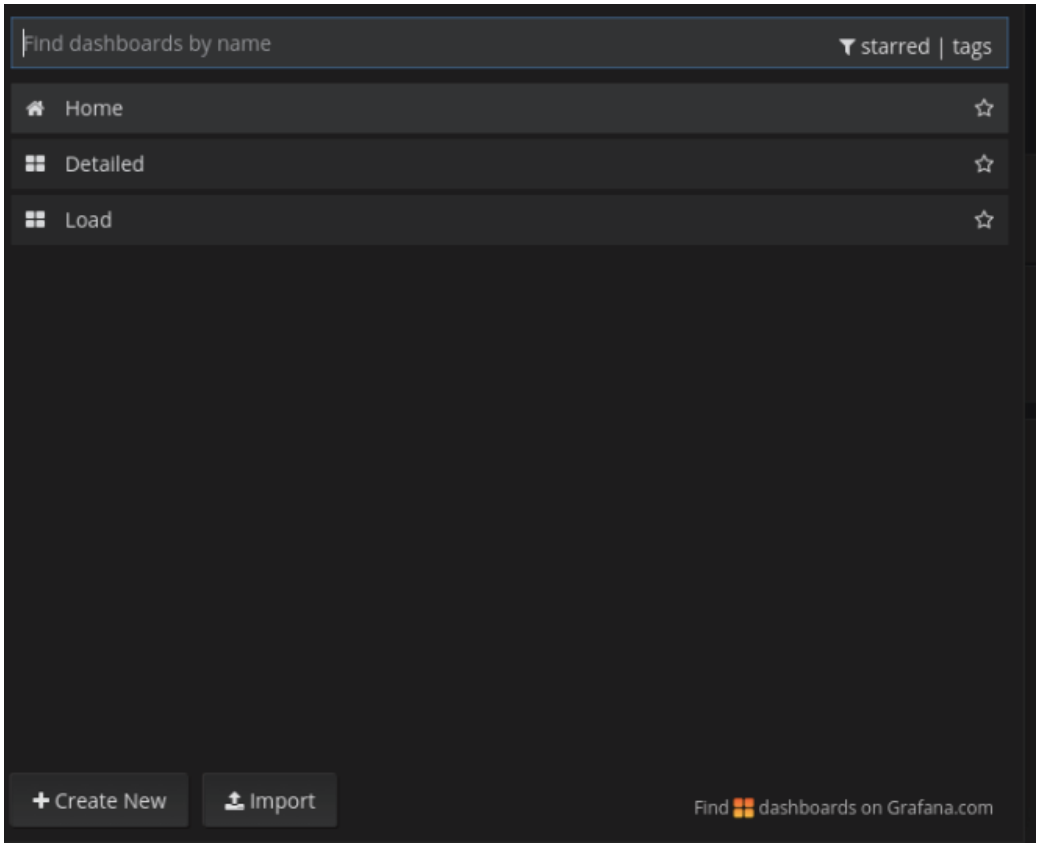

Рисунок. 4.5 – Сторінка імпорту інформаційних панелей

Рисунок. 4.6 – Вигляд завантаженої інформаційно панелі навантаження на ЦПУ сервера

4.3 Anomally detection як одна з можливостей побудованої системи моніторингу

Однією з основних функцій мови запитів Prometheus є агрегація даних, та їх подальша обробка у режимі реального часу. Такі можливості дають змогу ефективно використовувати систему моніторингу для виявлення аномалій.

Існує чотири ключові причини, чому важливо виявлятиити аномалії:

1) діагностування інцидентів;
2) виявлення регресій продуктивності програми;
3) визначення та вирішення зловживання;
4) безпека.

По-перше, дані часових рядів повинні бути агреговані правильно. Використаємо стандартний лічильник http_requests_total, що вказаний на рисунку 4.7, як джерело даних, хоча багато інших показників можна застосувати за допомогою тих же методів.

```
http_requests_total{
 job="apiserver",
 method="GET",
 controller="ProjectsController",
 status_code="200",
 environment="prod"
}
```

Рисунок. 4.7 – Функція агрегації даних

Як вкаано на рисунку 4.7, у нас є додаткові параметри: “method”, “controller”, “status_cod”, “environments”, а також параметри, які додає Прометеус, наприклад “instance” та “job”.

Далі ми повинні вибрати правильний рівень агрегації для даних, які ми використовуємо. Це називається проблема Goldilocks – “занадто багато”, “занадто мало” або “правильно” - це важливо для пошуку аномалій. Агрегувавши

дані із занадто великими проміжком часу, їх можна зменшити до занадто малих розмірів, створивши дві потенційні проблеми:

1) ми можемо пропустити справжні аномалії, оскільки агрегація приховує проблеми, які виникають у підмножинах наших даних;
2) якщо ми виявимо аномалію, важко віднести її до певної частини нашої системи без додаткового дослідження аномалії.

Але якщо агрегувати дані за занадто малий проміжок часу, може виникнути серія даних із дуже невеликими розмірами вибірки, що може призвести до помилкових результатів і може означати позначення підозрілих даних якправильних.

Виконаємо агрегацію всіх http запитів до серверів продуктового середовища часовим проміжком в п'ять хвилин. В результаті зробимо запис з виразом, конфігурація якого наведенна на рискунку 4.8:

```
- record: job:http_requests:rate5m
expr: sum without(instance, method, controller, status_code)
(rate(http_requests_total[5m]))
# --> job:http_requests:rate5m{job="apiserver", environment="prod"}  21321
# --> job:http_requests:rate5m{job="gitserver", environment="prod"}  2212
# --> job:http_requests:rate5m{job="webserver", environment="prod"}  53091
```

Рисунок. 4.8 – Конфігурація агрегації данних з п’ятихвилинним вікном

Деякі основні принципи статистики можуть бути застосовані для виявлення аномалій з Prometheus

Якщо ми знаємо середнє значення та стандартне відхилення (σ) ряду, ми можемо використовувати будь-яку вибірку в серії для обчислення z-score. Z-score вимірюється в кількості стандартних відхилень від середнього. Таким чином, z-score 0 означатиме, що z-score є ідентичною середній у наборі даних із нормальним розподілом, тоді як z-score 1 дорівнює 1.0 σ від середньої величини.

Якщо припустити, що базові дані мають нормальний розподіл, 99.7% зразків повинні мати z-score від нуля до трьох. Чим далі z-score дорівнює нулю,

тим менше ймовірність існування. Ми застосовуємо цю властивість для виявлення аномалій у серії Prometheus.

Обчислимо середнє та стандартне відхилення для показника, використовуючи дані з великим розміром вибірки. На рисунку 4.9 ми використовуємо дані, зібрані протягом одного тижня. При збиранні даних раз на хвилину, за тиждень у нас буде трохи більше 10 000 зразків.

```
# Long-term average value for the series
- record: job:http_requests:rate5m:avg_over_time_1w
expr: avg_over_time(job:http_requests:rate5m[1w])

# Long-term standard deviation for the series
- record: job:http_requests:rate5m:stddev_over_time_1w
expr: stddev_over_time(job:http_requests:rate5m[1w])
```

Рисунок. 4.9 – Збирання даних раз на 5 хвилин протягом тижня

Ми можемо обчислити z-score для запиту Prometheus, зображеного на рисунку 4.10, як тільки будемо мати середнє та стандартне відхилення для агрегації.

```
# Z-Score for aggregation
(
job:http_requests:rate5m -
job:http_requests:rate5m:avg_over_time_1w
) /  job:http_requests:rate5m:stddev_over_time_1w
```

Рисунок. 4.10 – Розрахунок z-score для агрегаціїї

Виходячи зі статистичних принципів нормальних розподілів, можна припустити, що будь-яке значення, яке виходить за межі діапазону приблизно від 3 до -3, є аномалією. Приклад виявлення таких аномалій зображено на рисунку 4.11:

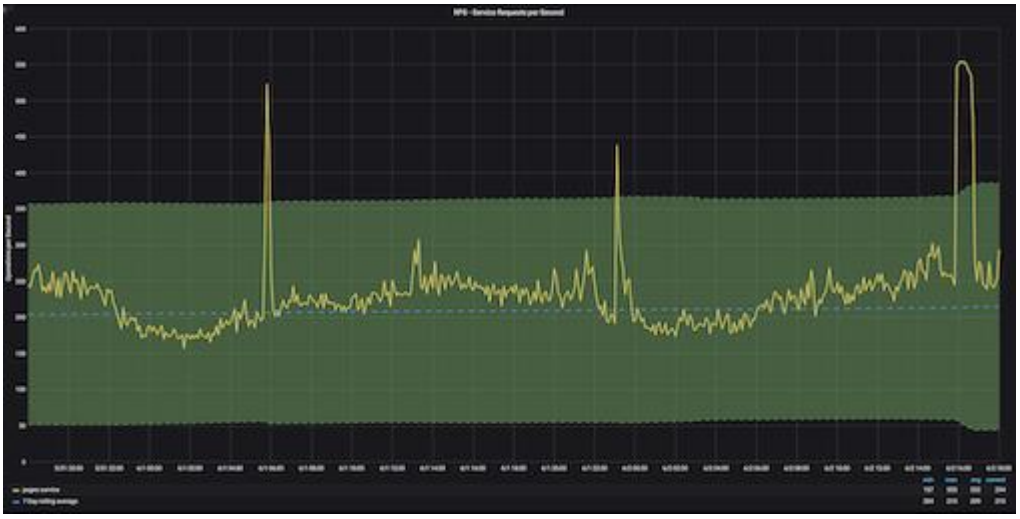

Рисунок. 4.11 – Виявлення аномалій поза межами інтервалу від 3 до -3

Z-score трохи незручно інтерпретувати на графіку, оскільки вони не мають одиниці вимірювання. Але аномалії на цій діаграмі легко виявити. Все, що з'являється за межами зеленої зони (що позначає z-score, які потрапляють в інтервал від +3 до -3), є аномалією.

Ми припускали, що наша основна агрегація має нормальний розподіл. Якщо ми обчислимо z-score за допомогою даних, які зазвичай не розподіляються, наші результати будуть неправильними.

Існує чимало статистичних методів тестування даних для нормального розподілу, але найкращим варіантом є перевірка того, що базові дані мають z-score приблизно від 4 до -4, що зображено на рисунку 4.12:

```
(
 max_over_time(job:http_requests:rate5m[1w]) - avg_over_time(job:http_requests:rate5m[1w])
) / stddev_over_time(job:http_requests:rate5m[1w])
# --> {job="apiserver", environment="prod"}  4.01
# --> {job="gitserver", environment="prod"}  3.96
# --> {job="webserver", environment="prod"}  2.96

(
 min_over_time(job:http_requests:rate5m[1w]) - avg_over_time(job:http_requests:rate5m[1w])
) / stddev_over_time(job:http_requests:rate5m[1w])
# --> {job="apiserver", environment="prod"}  -3.8
# --> {job="gitserver", environment="prod"}  -4.1
# --> {job="webserver", environment="prod"}  -3.2
```

Рисунок. 4.12 – Перевірка мінімального та максимального z-score

Показники, які, ймовірно, не мають нормального розподілу, включають такі показники, як частота помилок, затримки, довжина черги тощо.

Хоча обчислення z-score добре працює з нормальним розподілом даних часових рядів, існує інший метод, який може дати ще більш точні результати виявлення аномалії. Сезонність - характеристика метрики часових рядів, в якій метрика зазнає регулярних і передбачуваних змін, які повторюються кожен цикл. Графік значень запитів в секунду забражено на рисунку 4.13:

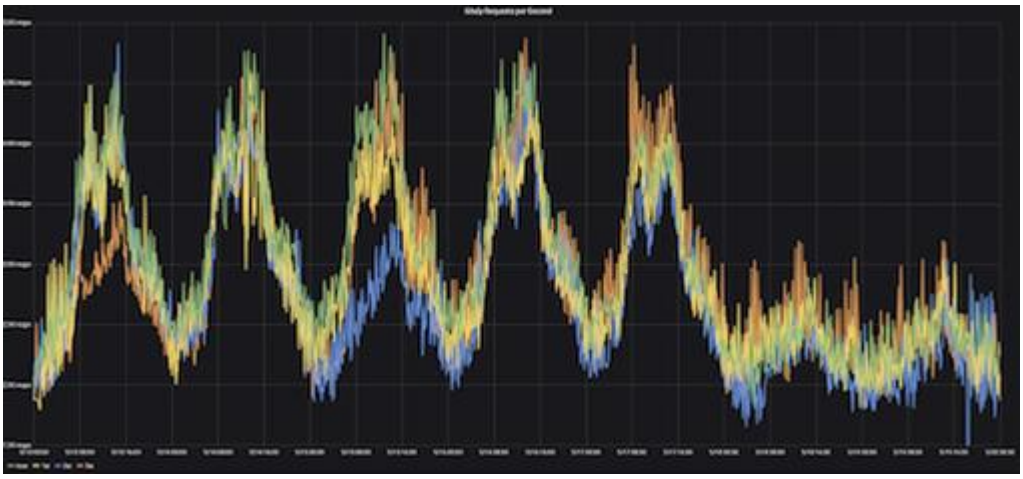

Рисунок. 4.13 – Графік RPS діапазоном в місяць

Цей графік ілюструє значення RPS (запити в секунду) протягом чотирьох тижнів поспіль. Семиденний діапазон називають «компенсуванням», що означає шаблон, який буде вимірюватися.

Кожен тиждень на графіку буває іншого кольору. Сезонність даних вказується на узгодженість тенденцій, зазначених на графіку - кожного ранку в понеділок ми спостерігаємо однаковий ріст значення RPS, а ввечері в п’ятницю ми спостерігаємо зниження значень RPS.

Використовуючи сезонність даних наших часових рядів, ми можемо створити більш точні прогнози, що призведе до ефективнішого виявлення аномалії.

Обчислення сезонності за допомогою Prometheus вимагало, щоб ми перейшли до декількох різних статистичних принципів.

У першій ітерації ми обчислюємо, додаючи тенденцію зростання, яку ми спостерігали за тижневий період, до значення попереднього тижня. Обчислимо тенденцію зростання, віднімаючи середнє однотижневе за останній тиждень від поточного середньотижневого на даний момент, що зображено на рисунку 4.14:

```
- record: job:http_requests:rate5m_prediction
  expr: >
    job:http_requests:rate5m offset 1w                    # Value from last period
    + job:http_requests:rate5m:avg_over_time_1w           # One-week growth trend
    - job:http_requests:rate5m:avg_over_time_1w offset 1w
```

Рисунок. 4.14 – Розрахунок тенденцій

Перша ітерація трохи вузька; ми використовуємо п'ятихвилинне вікно з цього тижня та попереднього тижня для отримання своїх прогнозів.

У другій ітерації ми розширюємо сферу застосування, беручи в середньому чотиригодинний період за попередній тиждень і порівнюючи його з поточним тижнім. Отже, якщо ми намагаємось передбачити значення показника о 8 годині ранку в понеділок, замість того, щоб використовувати одне і те ж п'ятихвилинне вікно за тиждень до цього, ми використовуємо середнє значення для показника від 6 ранку до 10 ранку для попереднього дня. Вираз такого розрахунку зображено на рисунку 4.15:

```
- record: job:http_requests:rate5m_prediction
  expr: >
    avg_over_time(job:http_requests:rate5m[4h] offset 166h) # Rounded value from last period
    + job:http_requests:rate5m:avg_over_time_1w             # Add 1w growth trend
    - job:http_requests:rate5m:avg_over_time_1w offset 1w
```

Рисунок. 4.15 – Розрахунок тенденцій

Ми використовуємо 166 годин у запиті замість одного тижня, тому що ми хочемо використовувати чотиригодинний період, виходячи з поточного часу доби, тому нам потрібно, щоб компенсація не містила дві години в тиждень.

На рисунку 4.16 зображено два графіки, реального та прогнозованого значення запитив за секунду інтерфалом в два тижні:

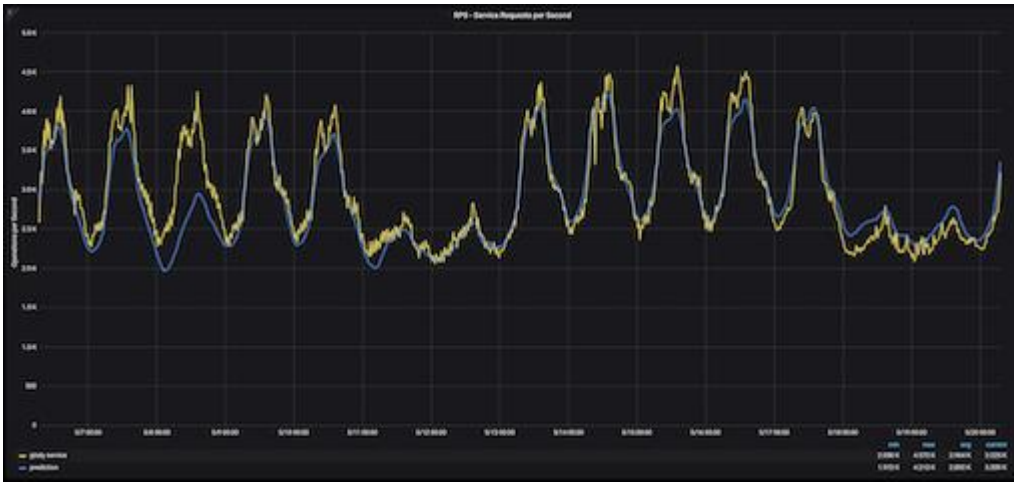

Рисунок. 4.16 – Реальне значення RPS (жовтий) та прогнозоване (синій), протягом двох тижнів

Одна з проблем цього підходу полягає в тому, що ми намагаємось включити три серії в агрегацію, а ці три серії - це фактично все ті ж серії протягом трьох тижнів. Іншими словами, всі вони мають однакові мітки, тому з'єднувати їх складно. Щоб уникнути плутанини, ми створюємо мітку під назвою зміщення та використовуємо функцію заміни мітки, щоб додати зміщення до кожного з трьох тижнів. Далі, в квантильній агрегації, ми розмічаємо це, і це дає нам середнє значення з трьох серій агрегацій, що зображено на рисунку 4.17:

```
- record: job:http_requests:rate5m_prediction
  expr: >
   quantile(0.5,
     label_replace(
       avg_over_time(job:http_requests:rate5m[4h] offset 166h)
       + job:http_requests:rate5m:avg_over_time_1w - job:http_requests:rate5m:avg_over_time_1w offset 1
w
       , "offset", "1w", "", "")
     or
     label_replace(
       avg_over_time(job:http_requests:rate5m[4h] offset 334h)
       + job:http_requests:rate5m:avg_over_time_1w - job:http_requests:rate5m:avg_over_time_1w offset 2
w
       , "offset", "2w", "", "")
     or
     label_replace(
       avg_over_time(job:http_requests:rate5m[4h] offset 502h)
       + job:http_requests:rate5m:avg_over_time_1w - job:http_requests:rate5m:avg_over_time_1w offset 3
w
       , "offset", "3w", "", "")
   )
   without (offset)
```

Рисунок. 4.17 – Розмічене середнє значення з трьох серій

Тепер наше прогнозування, яке зображено на рисунку 4.19, що виводить середнє значення з серії трьох агрегацій, набагато точніше.

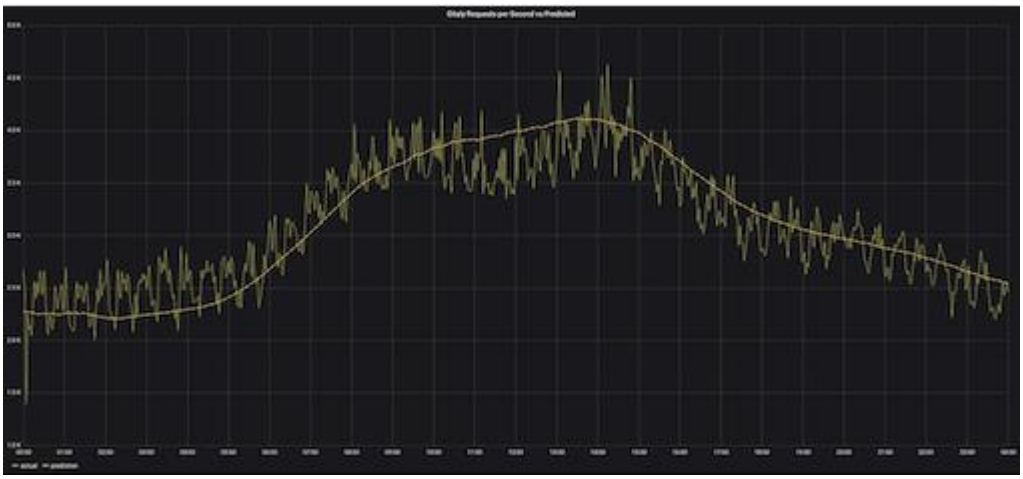

Рисунок. 4.19 – Графік агрегації з трьох серій

Щоб перевірити точність нашого прогнозування, ми можемо повернутися до z-score. Ми можемо використовувати z-score для вимірювання відстані вибірки від її прогнозування в стандартних відхиленнях, зображеної на рискнку 4.20. Чим більше стандартних відхилень від нашого прогнозування, тим більша ймовірність того, що певне значення є аномалією.

Рисунок. 4.20 – Використання z-score до агрегованого значення

Ми можемо оновити наш графік, щоб використовувати сезонне прогнозування, а не середнє значення щотижня. Діапазон нормальності для певного часу доби зафарбовується зеленим кольором. Все, що потрапляє за межі зеленої зони, вважається аномалією. У такому випадку аномалія була виявлена в неділю вдень.

Використання меж ± 2σ по обидві сторони нашого прогнозування є досить хорошим вимірюванням для визначення всплеску із сезонними прогнозами.

Ми можемо застосувати досить просте правило для AlertManager, яке перевіряє, чи є z-score метрики в діапазоні стандартного відхилення +2 або -2. Параметри сповіщення зображені на рисунку 4.21:

```
- alert: RequestRateOutsideNormalRange
  expr: >
   abs(
     (
       job:http_requests:rate5m - job:http_requests:rate5m_prediction
     ) / job:http_requests:rate5m:stddev_over_time_1w
   ) > 2
  for: 10m
  labels:
    severity: warning
  annotations:
    summary: Requests for job {{ $labels.job }} are outside of expected operating parameters
```

Рисунок. 4.21 – Налаштування сповіщення в разі виявлення аномалій

Ми використовуємо правило маршрутизації, яке надсилає сповіщення в Slack при виявленні будь-яких аномалій.

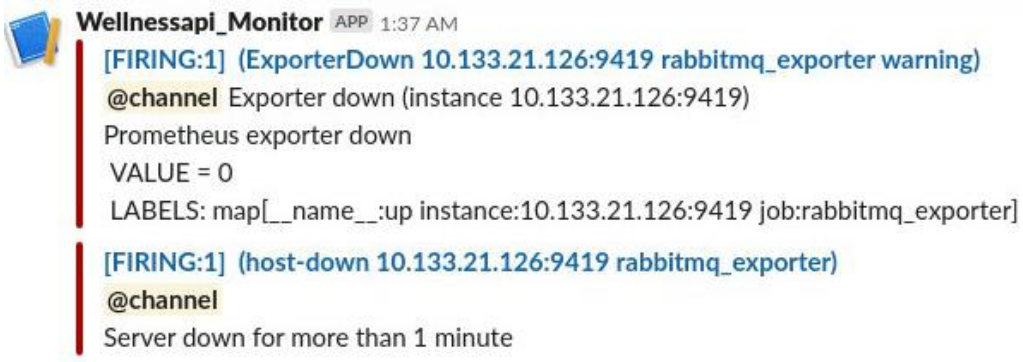

Рисунок. 4.22 – Приклад сповіщення в Slack

4.3 Висновки до розділу 4

В цьому розділі було наведено конфігурацію розробленої системи моніторингу на основі експертних систем та описано основні аспекти її налаштування, включаючи опис параметрів сценарію Ansible, встановлення та конфігурація Prometheus та AlertManager, а також налаштування засобу візуалізації Grafana.

Якщо розглядати побудовану систему моніторингу з боку експертних систем, то в ролі бази дних використовувався Prometheus експортер, бази знань та інтерпретатора – Prometheus, вирішувач – Prometheus AlertManager.

Для виявлення аномалій використовувалась агрегація даних за різний проміжок часу, а також встановлення “беспочного діапазону” за допомогою z-score. Побудована система моніторингу здатна самостійно адаптуватися до росту або падіння користувачів, постійного аналізу свіжих даних. Також варто зазначити що налаштована система сповіщень, та у випадку виявлення аномалій користувач отримує сповіщення в Slack.

РОЗДІЛ 5 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

У даному розділі буде розглянуто ключові особливості розробленої системи як майбутнього стартап-проекту. Проект розглядатиметься як система моніторингу на базі експертної системи.

5.1 Опис ідеї проекту

Спочатку проаналізуємо та подамо у вигляді таблиці зміст ідеї стартап-проекту, можливі напрямки застосування та основні вигоди, які може отримати користувач товару. Ці характеристики стартап-проекту зображено в таблиці 5.1.

Таблиця 5.1 - Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Набір програмного забезпечення з можливістю автоматизованого встановлення за гнучкого налаштування	Відслітковування стану серверної інфраструктури	Можливість швидко отримати актуальну інформації про стан серверної інфраструктури та встановлених на ній ПЗ. Поради щодо ефективної зміни конфігурації системи

Тепер зробимо аналіз потенційних техніко-економічних переваг ідеї порівняно із пропозиціями конкурентів. Результати аналізу зображено в таблиці 5.2.

Таблиця 5.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	Товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Конкурент 1	Конкурент 2			
1.	Ціна	2000$/рік	4000$/рік	5000$/рік			+
2.	Прибутки	30000$/рік	40000$/рік	20000$/рік		+	
3.	Контроль якості	Аналітики та програмісти	Аналітики, програмісти та деякі клієнти	Програмісти			+
4.	Динаміка галузі	Швидка	Повільна	Швидка		+	
5.	Постійні витрати	10000$/рік	20000$/рік	15000$/рік			+
6.	Змінні витрати	5000$ - 10000$/рік	1000$ - 2000$/рік	2000$ - 5000$/рік	+		
7.	Патенти на продукти	Немає	Патент на кожний проект	Декілька патентів на винахід	+		

Продовження таблиці 5.2

№ п/п	Техніко-економічні характери-тики ідеї	Товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Конку-рент 1	Конку-рент 2			
8.	Гнучкі ціни	Ціна єдина	Ціна варію-ється з року в рік	Ціна єдина		+	
9.	Законо-давчі обмеження	Обмеження на використання приватних даних користувачів – GDRP	Немає	Обме-ження на кіль-кість розро-бників			+

5.2 Технологічний аудит ідеї проєкту

Визначимо технологічну здійсненність ідеї проекту за допомогою аналізу таких складових, як технології, за якою буде виготовлено товар згідно ідеї проекту, існування таких технологій, чи їх необхідно розробити / доробити, доступність таких технологій авторам проекту. Результати даного аналізу зображено в таблиці 5.3.

Таблиця 5.3 – Технологічна здійсненність ідеї проекту

Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
Набір програмного забезпечення з можливістю автоматизованого встановлення за гнучкого налаштування	Система управління конфігурацією	Так	Дані технології доступні
	Time-series Data Base	Так	Дані технології доступні
	Система візуалізаї метрик	Так	Дані технології доступні.
Обрана технологія реалізації ідеї проекту: в реалізаї проекту будуть застосовані всі з наведених технологій.			

Проведемо аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку. Результати даного аналізу зображено в таблиці 5.4.

Таблиця 5.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	*Показники стану ринку (найменування)*	*Характеристика*
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж, грн/ум.од	300 000
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Висока точність розпізнавання, швидкодія, невибагливість до ресурсів
5	Специфічні вимоги до стандартизації та сертифікації	Немає
6	Середня норма рентабельності в галузі (або по ринку), %	80

Таким чином, за попереднім оцінюванням, ринок є привабливим для входження.

Надалі визначимо потенційні групи клієнтів, їх характеристики, та сформуємо орієнтовний перелік вимог до товару для кожної групи. Ці дані зображено в таблиці 5.5.

Таблиця 5.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп клієнтів*	*Вимоги споживачів до товару*
1	Потреба в ефективному моніторингу інфраструктури компанії	Будь-який ІТ проект, що має власну серверну інфраструктуру	Поведінка цільових груп однакова.	Клієнти прагнуть точності та швидкодії від системи. Ефективного сповіщення та мінімальної кількості хибних сповіщень.

Після визначення потенційних груп клієнтів проведемо аналіз ринкового середовища: складемо таблиці факторів, що сприяють ринковому впровадженню проекту (таблиця 5.6), та факторів, що йому перешкоджають (таблиця 5.7).

Таблиця 5.6 – Фактори загроз

№ п/п	*Фактор*	*Зміст загрози*	*Можлива реакція компанії*
1	Відсутність попиту	Бізнес може не оцінити переваги продукту, або ж у цілому відмовитися від ведення нарад	Акцентувати увагу на клієнтах, що вже скористалися продуктом, якщо такі є, навести інфографіку результативності (очікувану), запропонувати знижку потенційному клієнту в рамках тендеру.
2	Хибне сповіщення	Дефекти та особливості налаштування мережі, неточність експертної системи	Налаштування високодоступності системи, допрацювання експертної системи

Таблиця 5.7 – Фактори можливостей

№ п/п	*Фактор*	*Зміст можливості*	*Можлива реакція компанії*
1	Кобрендінг	Пропозиція від певної компанії, що спеціалізується на розробці моніторингових систем	Виділення частини штату на реалізацію проекту, підготовка акційних пропозицій по переходу на новий продукт існуючим клієнтам.

Надалі проведемо аналіз пропозиції: визначимо загальні риси конкуренції на ринку. Результати даного аналізу зображені в таблиці 5.8.

Таблиця 5.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	*В чому проявляється дана характеристика*	*Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)*
1. Чиста конкуренція	Гравці ринку не мають явних переваг один над одним	Більш вигідні умови на тендерах, агресивний маркетинг
2. Регіональна конкуренція	Гравці ринку – інтернаціональні підприємства	Вихід на ті ринки, які ще не зайняті конкурентами
3. Внутрішньогалузева конкуренція	Гравці ринку знаходяться в одній галузі – розробці ПЗ	
4. Товарно-видова конкуренція	Усі продукти гравців ринку мають одне призначення	Розробка найбільш інтуїтивного інтерфейсу, розробка унікальних мовленнєвих пакетів, оптимізація алгоритмів (щоб аналіз проходив швидше, ніж у конкурентів)
5. Конкурентні переваги нецінові	Продукти відрізняються гнучкістю, функціоналом (незначно) і надійністю.	У маркетингу неявно порівнювати власний продукт з іншими, робити вигідні цінові пропозиції
6. Марочна конкуренція	Значна увага приділяється бренду, що розробив продукт	Кобрендінг

Тепер визначимо та обґрунтуємо фактори конкурентоспроможності, які зображені в таблиці 5.9.

Таблиця 5.9 – Обґрунтування факторів конкурентоспроможності

№ п/п	*Фактор конкурентоспроможності*	*Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)*
1	Невибагливість до апаратних ресурсів (серверів). А отже дешевизна апаратних ресурсів, потрібних для нашої системи	В продукті використане поєднання технологій з відкритим вихідним кодом.
2	Швидкодія	В продукті використане поєднання систем контрою конфігурацій та систем моніторингу, що дозволить гарантувати швидке розгортання
3	Інтеграція	Продукт може бути використаний в будь-якій серверній інфрастріктурі.
4	Зручність	Замовник може обирати тип відслітковування стану серверів та ПЗ розгорнутих на них (за допомогою веб-сайту, мобільних додатків)
5	Гнучкість	Кожен замовник має можливість замовити розширення функціоналу продукту під його конкретні задачі

Таблиця 5.10 – Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Складові аналізу	Динаміка галузі, продуктова лінія, бар’єри проникнення	Наявність товарних знаків, доступ до ресурсів, патенти на продукти	Концентрація постачальників, диференціація витрат	Рівень чутливості до зміни цін, прибутки, контроль якості	Ціна, лояльність споживачів

Продовження таблиці 5.10

Виснов-ки:	Конкурен-ція не є інтенсив-ною, адже даний ринок ще ніким не зайнятий.	Для входу на ринок необхідно створити товарний знак та написати бета-версію програм-ного продукту. На даний момент потенцій-них конкурентів немає.	Постачаль-ники не диктують умови роботи на ринку, бо програмному продукту не потрібно постачання.	Клієнти диктують умови роботи на ринку, бо вони є єдиним джерелом прибутку компанії.	При наявності товарів замінників необхідно буде зменшува-ти ціну програм-ного продукту чи створю-вати ПЗ для інших технічних систем.

За визначеними факторами конкурентоспроможності проведемо аналіз сильних та слабких сторін стартап-проекту. Результати даного аналізу зображено в таблиці 5.11.

Таблиця 5.11 – Порівняльний аналіз сильних та слабких сторін системи «MonitOne»

№ п/п	*Фактор конкурентоспроможності*	*Бали 1-20*	*Рейтинг товарів-конкурентів у порівнянні з BioM*						
			–3	*–2*	*–1*	*0*	*+1*	*+2*	*+3*
1	Інтеграція	15					*		
2	Модульність	16			*				
3	Гнучкість	18			*				
4	Голосова аутентифікація	17			*				

Тепер проведемо SWOT-аналіз на основі виділених загроз і можливостей, та сильних і слабких сторін проекту. SWOT-матриця зображено в таблиці 5.12.

Таблиця 5.12 – SWOT-аналіз стартап-проекту

Сильні сторони: голосова аутентифікація, невибагливість до обчислювальних ресурсів, швидкодія	Слабкі сторони: Інтеграція має пройти із залученням розробників на стороні замовника
Можливості: Кобрендінг	Загрози: Неточність сповіщення, проблеми з мережею.

На основі SWOT-аналізу розробимо альтернативи ринкової поведінки для виведення стартап-проекту на ринок та орієнтований оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок. Дані альтернативи зображено в таблиці 5.13.

Таблиця 5.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	*Альтернатива (орієнтовний комплекс заходів) ринкової поведінки*	*Ймовірність отримання ресурсів*	*Строки реалізації*
1	Реалізація можливості використання системи не тільки на веб-сайтах, а й телефонних додатках	Середня	18 місяців
2	Створення системи негайного реагування	Висока	24 місяці
3	Розробка MVP	Висока	6 місяців

Серед даних альтернатив було обрано третю альтернативу, адже строки її реалізації найменші та є ймовірність отримання ресурсів.

Для розроблення ринкової стратегії першим кроком необхідно описати цільові груп потенційних споживачів, які можна побачити в таблиці 5.14.

Таблиця 5.14 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1.	Малий бізнес	Середня	5-10 підприємств в рік	Середня	Середня
2.	Середній бізнес	Готові	5-10 підприємств в рік	Слабка	Середня
3.	Великий бізнес	Готові	3-5 закладів в рік	Слабка	Складна
Було обрано цільову групу підприємств групи малого бізнеса.					

Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку, яку ображено в таблиці 5.15.

Таблиця 5.15 – Визначення базової стратегії розвитку

Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Концентрація на потребах одного цільового сегменту – веб-сайтах.	Створений продукт є дешевим у використанні та іноваційним	Стратегія спеціалізації.

Наступним кроком є вибір стратегії конкурентної поведінки, яку зображено в таблиці 5.16.

Таблиця 5.16 – Визначення базової стратегії конкурентної поведінки

Чи є проект «першопрохід-цем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
Так.	Компанія буде шукати нових споживачів, але і, за потреби, буде намагатися забирати існуючих у конкурентів.	Компанія, за потреби, буде копіювати характеристики конкурентів.	Стратегія заняття конкурентної ніші.

Тепер розробимо стратегію позиціонування, що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект. Її зображено в таблиці 5.17.

Таблиця 5.17 – Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспро-можні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
Реагування на зміни в інфраструктурі має бути точним Система є невибагливою до ресурсів Система є щвидкою	Проведен-ня крупних оновлень (оптимі-зація розрахун-ків), створення додатков-ого функціона-лу.	Товар є іноваційним (в тренді) та дешевим у використанні порівняно з альтернативами	Швидкий, невибагливий до ресурсів та зручний доступ до приватної панелі керування.

5.3 Розроблення маркетингової програми стартап-проекту

Сформуємо маркетингову концепцію товару, який отримає споживач. В таблиці 5.18 зображено результати попереднього аналізу конкурентоспроможності товару.

Таблиця 5.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1.	Швидке та точне сповіщення.	Виявлення проблеми в інфраструктурі та сповіщення про це відповідальної особи.	Менша ціна виявлення проблем.
2.	Невибагливість до апаратних ресурсів	Невибагливість до апаратних ресурсів клієнта	Доступність для компаній з невеликим капіталом.
3.	Швидкодія	Швидкодія системи	Більша швидкість

Надалі розробимо трирівневу маркетингову модель товару: уточнимо ідею продукту, його фізичні складові, особливості процесу його надання. Дана модель зображена в таблиці 5.19.

Таблиця 5.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
I. Товар за задумом	Програмний продукт – система моніторингу, яка дозволяє користувачу відслітковувати стан серверів в реальному часу, та своєчасно сповіщати відповідальну особу про виникнення будь-яких проблем
II. Товар у реальному виконанні	Властивості / характеристики: 1. Можливість індивідуально визначати конфігурацію системи в залежності від потреб клієнта 2. Можливість гнучко встановлювати методи сповіщення відповідальної особи 3. Можливість відслітковувати стан серверної інфраструктури
	Якість: програмний продукт пройшов всі етапі тестування та готовий до використання.
	Файли з розширенням “.yml”, віртуальне середовище.
	Марка: назва організації-розробника «EL», назва товару «MonitOne».
III. Товар із підкріпленням	Спеціаліст із впровадження встановлює ПЗ
	Відділ розробки підтримує життєдіяльність ПЗ

Тепер визначимо цінові межі, якими необхідно керуватись при встановленні ціни на потенційний товар, яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів. Аналіз проводився експертним методом і його результати зображено в таблиці 5.20.

Таблиця 5.20 – Визначення меж встановлення цін

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
3000-5000 $/рік	5000-6000 $/рік	12000-50000 $/рік	Нижня межа – 3000 $/рік, верхня межа - 5000 $/рік

Надалі визначимо оптимальну систему збуту, в межах якого приймається рішення. Дану систему зображено в таблиці 5.21.

Таблиця 5.21 – Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Клієнт виплачує гроші на рік, тоді до нього приходить спеціаліст із впровадження інформаційних систем і встановлює ПЗ на сервери клієнта.	Встановити програмний продукт на сервери клієнтів.	Один посередник – спеціаліст по впровадженню інформаційних систем.	Канал збуту одного рівня.

Тепер розробимо концепцію маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів. Дану концепцію зображено в таблиці 5.22.

Таблиця 5.22 – Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонува-ння	Завдання рекламного повідомлення	Концепція рекламного звернення

Продовження таблиці 5.22

Клієнт намагається знайти нові методи відслітковування стану серверної інфраструктури	Комп'ютерні мережі різного рівню, датацентри, хмарні провайдери	Швидкодія, невибагливість до апаратних ресурсів, іноваційність ПЗ, відносно невелика вартість ПЗ.	Продемонструва-ти швидкодію, дешевизну експлуатації, іноваційність та відносну невелику вартість ПЗ.	Показати можливість за невелику ціну зацікавити користувачі в свого продукту.

5.4 Висновки до розділу 5

В даному розділі було повністю виконано перший етап розроблення стартап-проекту, а саме, виконано маркетинговий аналіз стартап-проекту.

За допомогою нього можна сказати, що існує можливість ринкової комерціалізації проекту, адже на ринку програм систем моніторингу наявний попит на системи моніторингу що не вимагають багато ресурсів та впровадження яких є досить дешевими, до того ж рентабельність роботи є досить високою.

З огляду на потенційну групу клієнтів, а саме, ІТ-бізнес, що має власну серверну інфраструктуру є великі перспективи впровадження даного програмного забезпечення.

Для ринкової реалізації проекту доцільно обрати таку альтернативу впровадження: створення MVP та впровадження його в різні середовища.

ВИСНОВКИ

В цій дисертації були описані основні аспекти архітектури експертних систем, систем моніторингу та систем управління конфігурацій було розроблене та запропоноване рішення побудови експертної системи на основі інструментів моніторингу Prometheus – Grafana. Окрім того, розроблений сценарій автоматичного розгортання за допомогою засобу управління конфігурації Ansible.

При розв’язанні поставленої задачі вдалося розробити експертну систему, яка, окрім виконання обчислювальних операцій, здатна формувати певні висновки, базуючись на тих знаннях, якими вона володіє. Крім того, що основна кількість даних, що використовується для виявлення та запобігання збоїв, моніторинг на основі Prometheus - Grafana в свою чергу гарантує сповіщення відповідальної особи про актуальний стан IT-інфраструктури або її компонентів.

В цій роботі було запропоноване рішення для автоматизації розгортання експертної системи на основі інструментів моніторингу Prometheus - Grafana за допомогою засобу управління конфігураціями. Застосування саме експертних систем в сучасних системах моніторингу дозволить ефективно виявляти незакономірну роботу системи та запобігати збоям, які можуть бути викликані різними чинниками.

При втраті працездатності інформаційної системи чи виходу з ладу певних компонентів, засоби УК дозволяють з найменшими затратами ресурсів відновити коректну роботу ІС, за потреби провести централізоване оновлення інфраструктури, що може складатись з тисячі та десятків тисяч серверів.

Управління конфігурацією гарантує, що стан системи правильний, тож коли необхідно зробити розгортання та імплементацію достатньо виконання декількох команд, замість того, щоб розпочинати налаштування заново.

ПЕРЕЛІК ПОСИЛАНЬ

1. Chandola, V. Anomaly detection: A survey. *ACM Computing Surveys.* 2009. Vol. 41. No. 3. P 1–58.
2. Configuration Management process overview. *Micro Focus.* URL: https://docs.microfocus.com/SM/9.60/Codeless/Content/BestPracticesGuide_PD/ConfigurationManagementBestPractice/Configuration_Management_process_overview.htm
3. EJ HSU. Build A Monitoring Dashboard by Prometheus + Grafana. *DeepQ Research Engineering Blog.* URL: https://medium.com/htc-research-engineering-blog/build-a-monitoring-dashboard-by-prometheus-grafana-741a7d949ec2
4. Erika Heidi. An Introduction to Configuration Management. *DigitalOcean Community.* URL: https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management
5. Expert System in Artificial Intelligence: What is, Applications, Example. *Guru99*. URL: https://www.guru99.com/expert-systems-with-applications.html
6. IT Infrastructure Monitoring Essentials. *Heroix*. URL: https://go.heroix.com/it-infrastructure-monitoring
7. Victor García. Application Monitoring using ELK Stack. *Parque Científico de Madrid*. 2019. URL: https://www.enimbos.com/blog/en/application-monitoring-using-elk-stack/
8. What is Ansible? *Opensource.* URL: https://opensource.com/resources/what-ansible
9. What is Puppet? *Intellipaat.* URL: https://intellipaat.com/blog/what-is-puppet/
10. Експертні системи: особливості застосування. *Osvita.ua*. URL: https://osvita.ua/vnz/reports/management/13574/
11. Зачем нужен мониторинг ИТ-инфраструктуры и как его организовать. *Геолайн Технологии.* URL: https://geoline-tech.com/monitoring-it/

12.Основы мониторинга и сбора метрик. *8host.* URL: https://www.8host.com/blog/osnovy-monitoringa-i-sbora-metrik/

13. Субботін С. О. Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень: Навчальний посібник. Запоріжжя: ЗНТУ, 2008. 341 с.

14. What Is Prometheus? *Oreilly.*URL: https://www.oreilly.com/library/view/prometheus-up/9781492034131/ch01.html

15. Prometheus Annomally Detection. *A RED HAT BLOG.* URL: https://next.redhat.com/2019/11/18/prometheus-anomaly-detection/

ДОДАТОК А. ЛІСТИНГ ФАЙЛІВ КОНФІГУРАЦІЇ

GRAFANA-PROMETHEUS.YML

```
---
- include: prometheus.yml
- include: grafana.yml
```

PROMETHEUS.YML

```
---
- name: Set up prometheus
  hosts: prom-01
  gather_facts: true
  user: centos
  become: true
  vars_files:
  - ../group_vars/jhb.yml
  - ../group_vars/prometheus.yml

  roles:
    - prometheus
    - firewalld
```

GRAFANA.YML

```
---
- name: Set up grafana
  hosts: prom-01
  gather_facts: true
  user: centos
  become: true
  vars_files:
```

```
  - ../group_vars/jhb.yml

  roles:
    - grafana

tasks/main.yml
---
# tasks file for prometheus
- include: prometheus.yml
- include: alertmanager.yml

tasks/promtheus.yml
---
# tasks file for prometheus
- name: Create prometheus directory
  file:
    path: "{{ prometheus_dir }}"
    state: directory
    owner: root
    group: root
    mode: 0755

- name: Get prometheus
  unarchive:
    src: "{{ prometheus_link }}"
    dest: "{{ prometheus_dir }}"
    remote_src: True
    extra_opts: "--strip=1"

- name: Copy prometheus config
```

```
  template:
    src: prometheus.yml.j2
    dest: "{{ prometheus_dir }}/prometheus.yml"
    owner: root
    group: root
    mode: 0644
  notify: restart prometheus

- name: Copy rules file
  template:
    src: rules.j2
    dest: "{{ prometheus_dir }}/all.rules"
    owner: root
    group: root
    mode: 0644
  notify: restart prometheus

- name: Copy prometheus service file
  template:
    src: prometheus.service.j2
    dest: "{{ systemd_dir }}/prometheus.service"
    owner: root
    group: root
    mode: 0644
  notify: reload daemons
```

tasks/alertmanager.yml

```
---
# tasks file for prometheus
- name: Create alertmanager directory
  file:
```

```
    path: "{{ alertmanager_dir }}"
    state: directory
    owner: root
    group: root
    mode: 0755

- name: Get alertmanager
  unarchive:
    src: "{{ alertmanager_link }}"
    dest: "{{ alertmanager_dir }}"
    remote_src: True
    extra_opts: "--strip=1"

- name: Copy alertmanager config
  template:
    src: alertmanager.yml.j2
    dest: "{{ alertmanager_dir }}/alertmanager.yml"
    owner: root
    group: root
    mode: 0600
  notify: restart alertmanager

- name: Copy alertmanager service file
  template:
    src: alertmanager.service.j2
    dest: "{{ systemd_dir }}/alertmanager.service"
    owner: root
    group: root
    mode: 0644
  notify: reload daemons
```

templates/prometheus.yml.j2

```
global:
  scrape_interval:     {{ scrape }}
  evaluation_interval: {{ evaluation }}

rule_files:
  - all.rules

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
        labels:
          alias: prometheus
{% for item in prometheus_hosts %}
  - job_name: '{{ item }}'
    static_configs:
      - targets: ['{{ item }}:9100']
        labels:
          alias: {{ item }}
{% endfor %}
```

templates/alertmanager.yml.j2

```
global:
  # The smarthost and SMTP sender used for mail notifications.
  smtp_smarthost: '{{ smtp_server }}:{{ smtp_port }}'
  smtp_from: '{{ from_address }}'
  smtp_auth_username: '{{ smtp_username }}'
  smtp_auth_password: {{ smtp_pass }}
```

```
route:
  receiver: default
  group_interval: 0s

receivers:
  - name: default
    email_configs:
{% for item in send_to %}
      - to: '{{ item }}'
{% endfor %}

templates/rules.j2
{% raw %}
ALERT service_down
  IF up == 0
  ANNOTATIONS {
    summary = "{{ $labels.job }} is down",
  }

ALERT cpu_threshold_exceeded
  IF (1 - avg by(job)(irate(node_cpu{mode='idle'}[5m]))) > .90
  ANNOTATIONS {
    summary = "{{ $labels.job }}'s CPU usage is dangerously high",
    description = "{{ $labels.job }}'s CPU usage has exceeded the 90% threshold with
a 5 minute load value of {{ $value }}.",
  }

ALERT mem_threshold_exceeded
  IF ((node_memory_MemTotal - node_memory_MemFree - node_memory_Cached)
/ (node_memory_MemTotal )) * 100  > 90
```

```
  ANNOTATIONS {
    summary = "{{ $labels.job }}'s memory usage is dangerously high",
    description = "{{ $labels.job }}'s memory usage has exceeded the 90% threshold with a value of {{ $value }}%.",
  }

ALERT filesystem_threshold_exceeded
  IF node_filesystem_avail{mountpoint='/'} / node_filesystem_size * 100 < 20
  ANNOTATIONS {
    summary = "{{ $labels.job }}'s filesystem usage is dangerously high",
    description = "{{ $labels.job }}'s filesystem only has {{ $value }}% free.",
  }
```

{% endraw %}

templates/prometheus.service.j2

```
[Unit]
Description=Prometheus server

[Service]
Type=simple
ExecStart={{ prometheus_dir }}/prometheus -config.file {{ prometheus_dir }}/prometheus.yml -alertmanager.url=http://{{ alertmanager_address }}:9093 -storage.local.retention=168h

[Install]
WantedBy=default.target
```

templates/alertmanager.service.j2

```
[Unit]
Description=Prometheus Alertmanager

[Service]
Type=simple
ExecStart={{ alertmanager_dir }}/alertmanager -config.file={{ alertmanager_dir }}/alertmanager.yml

[Install]
WantedBy=default.target
```

vars/main.yml

```
# vars file for prometheus

prometheus_link:
https://github.com/prometheus/prometheus/releases/download/v1.6.2/prometheus-1.6.2.linux-amd64.tar.gz
alertmanager_link:
https://github.com/prometheus/alertmanager/releases/download/v0.6.2/alertmanager-0.6.2.linux-amd64.tar.gz

prometheus_dir: /opt/prometheus
alertmanager_dir: /opt/alertmanager
systemd_dir: /usr/lib/systemd/system

alertmanager_address: localhost

scrape: 5s
evaluation: 5s
```

```
smtp_server: smtp.gmail.com
smtp_port: 587
from_address: <your from address>
smtp_username: <your email username>
smtp_pass: !vault |
        $ANSIBLE_VAULT;1.1;AES256
```

handlers/main.yml

```
---
# handlers file for prometheus
- name: reload daemons
  shell: systemctl daemon-reload

- name: restart prometheus
  service:
    name: prometheus
    state: restarted

- name: restart alertmanager
  service:
    name: alertmanager
    state: restarted
```

tasks/main.yml

```
---
# tasks file for grafana
- name: Install grafana
  package:
```

```
    name: "{{ grafana_pkg }}"
    state: present

- name: Start grafana
  service:
    name: grafana-server
    state: started
    enabled: yes
```

handlers/main.yml

```
# handlers file for grafana
- name: restart firewalld
  service:
    name: firewalld
    state: restarted
```

vars/main.yml

```
---
# vars file for grafana
grafana_pkg:    https://s3-us-west-2.amazonaws.com/grafana-releases/release/grafana-4.2.0-1.x86_64.rpm

grafana_port: 3000
```